A BRANCH AND BOUND SOLUTION OF JOB-SHOP SCHEDULING PROBLEM USING GNATT-CHARTS

A Thesis submitted
in partial fulfilment of the requirements
for the degree of
Master of Technology

by

Jai Prakash Agrawal

Department of Electrical Engineering
Indian Institute of Technology
Kanpur

June 1970

CERTIFICATE

This is to certify that this work on "A Branch and Bound Solution of Job-Shop Scheduling Problem Using Gnatt-Charts" has been carried out under my supervision and it has not been submitted elsewhere for a degree.

V. Rajaraman
Professor
Department of Electrical Engineering

ACKNOWLEDGEMENT

The author wishes to express his deep sense of gratitude to Dr. V. Rajaraman for suggesting a challenging problem, the valuable guidance and the thought-provoking discussions.

SYNOPSIS

This thesis deals with a general mxn Job-Shop Scheduling problem with an objective to minimise the sum of the costs of job late-times and machine - idle-times. An optimal procedure using the branch and bound technique in which the tree of sequences is constructed on the basis of clashes in the Gnatt-chart of the problem has been evolved. Also a suboptimal procedure, based on the proposed technique, has been suggested to replace the thumb and heuristic rules used hitherto.

TABLE OF CONTENTS

CHAPTER I

THE JOB-SHOP SCHEDULING PROBLEM

In recent years there has been a great interest in Industrial scheduling problems from the point of view of both practice and research. Scheduling is the managerial decision of fixing the sequence of some activity (job) on some facility (machine) with respect to other activities (jobs) optimising certain cost or profit function. In this introductory chapter an attempt is made to enumerate the vast dimensions of the above problem.

A job-shop in any industry usually consists of following :

(1) A number of various kinds of machines, each kind of machine may be available in more than one number.

(2) The raw material, its storage, arrival and distribution mechanisms.

(3) The human resources and their efficiencies.

(4) The storage and despatching mechanisms for the finished products.

The jobs, to be processed in the above job-shop usually have the following specifications :

(1) The sequence of machines to be used by the jobs.

(2) The processing times of jobs on various machines.

(3) Cost function of the completion of the jobs.

(4) Arrival pattern of jobs (or job orders).

The object of the floor manager is to manage the job-shop most economically. Inclusion of all the above factors and a few more makes a problem of managing the complete industry. However, the job-shop-problem, as it is generally referred to, is that of assigning jobs on various machines. Before describing the job-shop problem in the present context in greater detail, let us see the kind of the cost functions associated with this problem.

The cost functions are the result of the interaction of the shop management and the customers. Following may be the usual customer's demands :

(1) The customer asks for the finished job on a due date. If the job is late the penalty as a function of the delay is specified. This function may be nonlinear. Early dispatch may or may not be favoured.

(2) The customer wants the job at a certain date, whatever the cost be. If it cannot be given he might like to scrap the deal. No early despatch may also be specified.

(3) A customer wants his job to cost least no matter when it is completed.

(4) Another customer wants his job to be rushed. They may be prepared to reward early despatch.

Many other situations or combinations of them can occur. Being faced with varying demands like above, the shop management has to assess their capacity. They have to prepare the tentative schedules which together with the above demands also incorporates the cost of machine idle times, and inprocess job inventories. After accounting for the allowances for mishaps, failures etc. the management should be able to say whether it can make them or not. If he can, then he should be able to specify the costs incurred and have a diologue with the customer. All the above cost requirements are nonlinear and therefore requires nonlinear programming techniques for getting an optimal solution.

Besides the complex nonlinear optimizing function the constraints on the job-shop also pose a serious problem. Let us enumerate them as follows :

(1) No machine may process more than one operation at a time.

(2) Certain operations on the jobs may be interruptible and others not.

(3) Assembly or diassembly of some jobs.

(4) Dynamic or batch input of jobs.

(5) Time bounds for the gap between two operations of a job like the usual examples in chemical processing.

(6) Uncertainty in the processing times given by the customer, or estimated elsewhere.

(7) One or more machines of one kind.

(8) Alternate sequence of processing on jobs, i.e., alternate routing of jobs.

(9) A job may be abandoned if the order is cancelled.

(10) A job may or may not be processed by more than one machine.

(11) Some jobs may not have rigid routings.

The list is not exhaustice but only illustrative of how complex the practical problems can be. There exists no analytical model to consider the above mentioned general job-shop scheduling. Even the simplest problems consisting of five to six machines and the same order of jobs require a large computer time to obtain the optimal solution. The only recourse left is to use heuristic rules and simulate the job-shop. It requires considerable experimentation.

In this thesis an optimal procedure, based on the branch and bound technique, has been developed for the solution of a general job-shop scheduling. The job-shop considered is as follows :

(1) Batch input of jobs.

(2) All machines are of different kinds.

(3) The processing times are known apriori.

(4) The costs are linear functions of joblateness and the machine-idle times.

(5) No alternate routes.

(6) No assembly or disassembly and no interruptions of any operations.

Later a sub-optimal procedure is given based on the above procedure. It can be substituted for any heuristic rules of scheduling in use so far. In the last some suggestions are given for taking care of the nonlinear cost functions, i.e., due dates, interruptions and more than one machine of one kind.

CHAPTER II

PRESENT METHODS OF SOLUTION

Knowing the complex structure of a general job shop problem, no one has tried to solve it completely. Many simplified models have been put forward and analysed. Frequently considered simplification are listed below :

(1) No machine may process more than one operation at a time.

(2) No operation on any job is interruptible and no job requires processing by a machine more than once.

(3) No assembly or disassembly allowed.

(4) The processing times include the set up times and are fixed.

(5) There are no constraints on the gap between two operations on a job.

(6) No alternate routes.

(7) There is only one of each type of machine.

(8) There is batch input of jobs.

The optimizing function has also been simplified, and the ones frequently considered are :

(a) Minimizing the make span time or the time of completion of all processes on the jobs.

(b) Minimizing the idle time of the machines.

The less frequently considered optimizing functions are :

(c) The cost of lateness of jobs and idle time on machines together.

(d) Meeting the due dates of the jobs.

Besides considering the simpler problems, the size of the problems considered are also small. For a m jobs n machines there exist $(m!)^n$ number of possible combinations. This number is around a million for a six by six problem. Both analytical and heuristic rules have been deployed to find optimal and sub-optimal solutions, which are reviewed in following articles.

2.1 MONTE CARLO TECHNIQUES

Despite the vastness of the problem Giffler et al(1) report on computational experience with the complete enumeration of all feasible schedules. According to them it need search only the active feasible schedules which are defined as follows :

An active schedule is one in which both job and machine are not simultaneously idle.

Such schedules can be generated by assigning a job to a machine at an earliest time at which both the job and the machine are free.

It reduces considerably the number of feasible schedules to be searched. Note that the active scheduling is dependent on the distribution of the processing times and routings on different jobs. Hence it cannot be calculated in advance. Giffler suggests Monte Carlo sampling of the Active feasible schedules. In this

technique queues are formed on the machines. If conflict occurs at any machine it is resolved by random choice. They observed that the short schedules have higher probability in the Monte Carlo distribution than in the true distribution obtained by linear complete enumeration algorithms. The reason propounded by Giffler et al. is that short schedules have, on the average fewer conflicts to be resolved than long ones. Since in the Monte Carlo algorithm each conflict is resolved at random and therefore, the probability of obtaining a particular schedule is approximately proportional to the number of resolved conflicts it has. Hence the longer schedules are less probable. After experimentation with a large number of problems they drew the following conclusions :

1) A Monte Carlo process that uses some heuristic rules as guides in its random choice, will be considerably superior to a purely random choice. The "shortest imminent operation" (SIO) has been mentioned to be a good choice for such a rule. In this rule the operator assigns that operation to the machine which will be completed first.

2) The larger the ratio of the longest to shortest job time (the shortest job time being 1), the larger the combinatorial aspects ofthe problem and therefore more difficult it is to schedule.

2.2 SIMPLE LOADING AND HEURISTIC RULES

Arbitrary loading rules have so far been the only methods in use in the industrial scheduling situations. Following are some of these rules to pick up the next job from a queue for scheduling :

(1) First come first serve

(2) Job with the shortest completion time

(3) Job that has the maximum number of uncompleted operations still to be performed, etc.

As against these rules, some rules are devised based on the skill and the experience of the schedulers. These are termed heuristic rules. Some of them are as follows :

(1) SIO : Choose that operation for next scheduling which can be completed in shortest time.

(2) LRT : Choose that job for next scheduling which has longest remaining time for completion.

(3) JOB SLACK : Choose that job for the next scheduling which had so far the largest ratio of slack time to operation. This ratio is calculated by dividing the slack time on a job by the number of operations so far scheduled on that job.

(4) Expected delay time : In this due-date problem the total delay in any job is estimated heuristically on the basis of the delay already occured on that job. The job with the largest

delay time and its associated cost, is chosen to be scheduled next.

(5) JOB SLACK Ratio : It is calculated by dividing job slack hours by hours remaining until the due date. The job with largest ratio is chosen for next scheduling.

V. Gere (2) has studied the performances of all the above rules together with random selections rule for the "due-date" problem. After selecting an operation according to some rule, he uses the following enhancement schemes :

1) Alternate Operation : ... Schedule the operation according to the rule. Now check to see if this makes another job 'critical' (that is to see if the slack of any other job has just become negative, or if positive, has reached a certain critical level). If so, revoke the last operation and schedule the next operation on the critical job. Check again for lateness. If scheduling the second operation does not cause any job to be critical, whereas the first one did, then schedule the second one, otherwise schedule the first one (that is one which was dictated by the rule).

2) Look ahead : ... When an operation is scheduled, we may ask : Is there a critical (i.e. late or nearly late) job due to reach this machine at some near future hour, yet before the scheduled operation is completed? If so, schedule that job. Check to see the effect of this on other jobs. Either let the schedule remain, or replace the operation with the previously scheduled operation, depending upon the resulting job lateness effected ...

3) Insert : Once a "look ahead" job has been scheduled there is a period of idle time on the machine, starting at the 'present time'. If there is a job in the queue whose next operation can be completed by the time the look ahead job is due to arrive at this machine. then the operation should be scheduled....

Gere's results are very interesting :

(1) The selection of a priority rule for discriminating between jobs competing for time on the machines, is not as important as the selection of a set of proper heuristics (i.e., the enhancement schemes).

(2) The SIO and job slack ratio are the simplest and most effective rules.

The heuristic enhancement schemes proposed by Gere are complex. However it is unavoidable because of the combinatorial nature of the problem and the nonlinearity of the cost function.

There is a large list of authors who proposed different heuristic rules. Efficiency of these rules depend upon particular kind of the problems they have worked on.

The heuristic rules do give quick solutions. However, for different optimising functions, different heuristic rules have to be invented. No author has discussed some generalised procedure to develop heuristic rules for different optimising functions.

An attempt in the above direction was made by the author of this thesis. The scheduling of an operation can be decided by an index which is calculated on the basis of the following parameters :

(1) Average time of operation on a job.

(2) Number of remaining operations on a job.

(3) The duration of the 'present' operation being considered for the decision of scheduling.

A search was made by giving different weights to these parameters. The weight thus obtained for different costs on the same problem and different problems were compared. There seemed to be least correlation among the results. The reason for this is surely the combinatorial structure of the problem. This attempt was abondoned in favour of the hope to be able to find such a heuristic procedure from the Branch and Bound technique which assures an optimal solution. This portion is dealt in the fourth chapter.

2.3 LEARNING TECHNIQUES

The authors working on the heuristic procedures of solution found that some rule works better in one case than others, in another case the reverse may happen. This prompted Fisher and Thompson (3) to use more than one such rule to construct a schedule.

They have usedMonte Carlo random scheduling by varying the probabilities of selecting specific decision rules at any stage of the scheduling process. The variation is dependent upon the history of actions and results encountered by previous decisions at the same point. The actions are the rule choices; the result is the make span time, the time of completion of the whole set of jobs. The probability of selecting a rule is varied on the basis of the comparison between the make span time of the present schedule with respect to the previous schedule or some previously established best schedule. Fisher et.al. have used SIO and LRT rules for their experimentation. They used many different heuristic devices for improving upon the probabilities of selection of any rule. They conclude that learning is possible and the learnt combinations of two or more rules gives a schedule better than the best that can be generated using the rules singly. They have also observed that using SIO more frequently in the beginning and LRT in the end of the scheduling process is a good combination.

It has the effect of getting machines to work quickly in the beginning and later concentrating on the long jobs.

Both the amount and the speed of learning depends strongly upon our ability to find good heuristic rules to start with. So here again the problem comes to finding rules suited to the optimising function of the problem.

An alternative to the learning using different rules as choices, can be the use of a weighted index as mentioned in the previous article. The learning can be done by assigning probabilistic weights to various parameters. The bottleneck with all types of learning is that it has to be done afresh on each problem.

2,4 BRANCH AND BOUND TECHNIQUES

This is a search technique which aims at finding a way (or ways) for the early discarding of some classes of permutations which are suspect of being 'bad'. It has been used to find the minimum make span time of a set of jobs. Most of the work in this direction is limited to the problems in which all the jobs have identical sequence of processing on various machines.

Lomnicki and Brown (4) and (5) have used this technique to find the minimum make span time for the type of problem mentioned above. They have used Roy's result which says that "in searching for optimum solution (to the make span problem) we can limit ourselves to the investigation of those schedules which have an identical order of jobs on the first two machines and an identical order of jobs on the last two machines." It reduces the number of permutations from $(n!)^m$ to $(n!)^{m-2}$ where n and m are respectively the number of jobs and machines. For the 3-machine case Lomnicki has given the following procedure: (1) Calculate the bounds on the make span by considering

each of the given jobs to be scheduled first. These bounds represents the nodes. Choose the job with the lowest bound and mark it.

(2) Calculate bounds for deciding the next position from the remaining jobs. The lowest bound of these and the unmarked nodes of all previous steps, is chosen and marked. Further decisions are taken from this node onward.

This procedure is continued until lowest of all the unmarked nodes is found, which gives the complete schedule.

Ability of getting a good lower bounds is the crux of the success of this method. Brown and Lomnicki propose the following bounds :

"G_j : total time of processing of all the jobs on machine j plus the shortest time of processing a job (say ith job) on machines 1,2 ...(j-1) plus the shortest time of processing a job (say 1th job) on machines (j 1), ... m, where clearly the job i must be different from job 1".

For the three-machines case Lomnicki have used yet another result from Roy's thesis that reversing the order of machines has as a solution a permutation in which the order of jobs is inverted. Solution of inverted problem is easier in some cases.

After generalising their method to general m machines, they report that the greater the difference between the initial lower bound and the optimum work duration the more

difficult it is, on the average, to evaluate the optimal sequencing. This is the problem of backtracking in the solution tree.

In their paper they have not shown how the generalisation to m machines will give the optimal result. From Roy's first conclusion the number of permutations to be searched are $(n!)^{m-2}$, whereas the authors have investigated only n! schedules. The remaining permutations are those in which the orders of processing of jobs on intermediate machines are not identical. Later Ignall and Schrage have mentioned an example to this effect. However, he tells that restricting ourselves to identical orders will give a "good schedule". Such permutations cannot be accounted if we take a job as an entity for the scheduling decision. We must find a branch and bound procedure which is based on scheduling the individual operations.

Ignall and Schrage(6) have given a branch and bound procedure very similar to the one given by Brown and Lomnicki except that certain branches are closed for further downward branching by performing dominance checks. These checks are used when backtracking becomes necessary. They argue that if two partial sequences involve identical jobs, then the sequence with the larger bound need not be considered anymore. Later Dudek and Smith, Bagga and Chakraberti (17) and G.B.Mcmohan (16) have improved upon

these dominance checks to be applied to two partial sequences in which only a few scheduled jobs are in common. Ironically these checks depend very strongly on the identical sequences of processing on all jobs.

Dudek and Teuton (7) came out with another formulation but Karush has given a counter example for it. D.S. Palmer (8) has given a near optimal procedure of getting an optimum sequence by a slope-order index of discrimination. The index, if converted in non-mathematical language, says that "give priority to the items having the strongest tendency to progress from short times to long times in the sequence of process". Palmer has limited his technique to problems of finding make span time in which all the jobs have identical orders of processing.

Some improvements in the lower bounds as given by Brown and Lomnicki and others, have been proposed by J.N.D. Gupta (15) in his recent paper. He reports 5 to 30% gain in reducing the computation time.

2.5 INTEGER LINEAR PROGRAMMING SOLUTIONS

Storey and Wagner has done the pilot study in this field. They introduce binary valued variables x_{ij} which are given a value 1 if item i is scheduled in order-position j and zero otherwise. Number of such variables is quite large even for small size problems.

They start with an optimal simplex solution and add the integer constraints one by one till the integer solution is found. They considered various schemes for selecting the basic variables and its corresponding equation on which to form the new constraint :

(1) Select a basic variable having the largest fractional part.

(2) Select an x_{ij} closest to 0.5.

(3) Select an $x_{ij} < 1$ closest to 1.

(4) Select an $x_{ij} > 0$ closest to 0.

(5) Select an $x_{ij} < 1$ closest to 1; whenever the objective function has a fractional part, force the value of the objective function to the next integer level and so on.

They observe that rule 3 works better than rule 2. They contended that they have not yet found an integer programming method that can be relied upon to solve the problems rapidly.

The formulation of a generalised mxn job-shop, given by Greenberg(10), is one of the leading steps in obtaining analytical solutions to this problem. He has used the branch and bound technique as given by Land and Doig (11) to solve the linear Integer programming version of the problem. Going down the tree he fixes the sequence between two jobs on a machine. It is equivalent to assigning a binary valued variable zero or one. The bounds are

the evaluation of the cost function. A detailed analysis of this procedure is given in Chapter III.

Egon Balas has proposed an implicit enumeration algorithm for solving integer linear programming problem (LPP) in which certain branches are blocked on the basis of cost in tests. The domaninance checks used by Ignall and Schrage and their followers also do essentially the same thing. The integer LPP formulations, as given by Balas, is identical with that of Greenberg's. He splits the original LPP in two. The first one actually optimizes the cost, while the second one chooses an optimum 0-1 variable (a sequence between two jobs on a machine) out of a set of unassigned binary variables (i.e., unresolved sequences). This variable is assigned a value 0 or 1 and inserted in the first part which is then optimised. Balas has simplified this procedure for solving the make span problem. He starts with a heuristically generated schedule. He chooses to reverse the sequence between two jobs on a machine which lie adjacent on the longest path of the schedule. The resulting schedule is tested to find whether we can say that no further decrease in the make span time will take place by the reversal of any new sequence in the schedule. If the tests assure it, then a backtracking step is taken and a new branch is explored corresponding to reversal of another sequence

on the longest path of that schedule. A backtracking step that brings the scheduler to the root of the generated tree assures the optimality of the solution from where the present backtracking step was originated. This is a post-optimization process. All depends upon getting a suitable heuristically generated initial schedule.

Nicholson (14) has developed a near optimal procedure to solve the job-shop problem with due dates and priorities. His procedure employs a thinking which is very similar to that of Balas. He also prepares an initial schedule and then reverses the sequence between the operations one by one until a lowest value of cost is found. The sub-optimality is introduced by limiting the sequence - reversals. He has reported considerable improvement over the heuristic and thumb rules.

All the above analytical techniques are still bagged in the elementary problems. The future of branch and bound procedure is very good, because as against Balas's method it gives a procedure to construct the schedule.

CHAPTER III

BRANCH AND BOUND METHOD

The technique to solve the integer linear programming problem was originally put forward by Land and Doig. Roy, Lomnicki and few others have used it to solve the job shop scheduling with two or three machines. The most general formulation of the problem with m jobs and n machines has been given by Greenberg. This formulation is discussed briefly in the first two articles of this chapter. The third article proves that making a Gnatt-chart is equivalent to solving a linear programming problem (LPP) used by Greenberg to find the schedule at any stage of the branch and bound procedure. The fourth article enumerates some of the limitations of Greenberg's method. In the fifth article an alternate procedure is developed on the lines of the Greenberg's formulations. The latter article gives the bounding procedure for the developed method.

3.1 PROBLEM STATEMENT

Given m jobs and n different kinds of machines, each job is independent and is required to be processed on n machines in a fixed sequence, a schedule is to be found which is optimal for a given linear objective function.

The restriction of n processes on each job can be dropped, but for simplicity in presentation of the main

iders. The linear objective functions can be enumerated as follows :

(a) Minimising the time of completion of all the jobs, referred to as minimising the make span.

(b) Minimising the idle time on the machines.

(c) Given costs of job idle times and machine idle times which vary linearly with the idle times, minimise the total costs of the schedule.

The notations used are the same as given by Greenberg. Let the sequence of machines for job i be N_i $(i=1,2,...m)$ and the kth element of this vector be $l(i,k)$. Let the start times S_{ik} denote the start time of job i on the machine k. The durations D_{ik} denote the processing time of job i on machine k.

Then the constraints of sequence of processes on any job can be written as follows :

$$S_{il(i,k)} + D_{il(i,k)} \leq S_{il(i,k+1)} \qquad (1)$$

$(k = 1.2... M-1)$ for all i.

Since only one machine of each kind is available, the jobs will have to compete for getting a time slot on any machine.

Let two jobs i and j be competing for a time slot on machine k, then if job j precedes (denoted by $<$) job i then,

$$S_{ik} \geq S_{jk} + D_{jk} \qquad (2)$$

and if job i < job j, then

$$S_{jk} \geq S_{ik} + D_{ik} \qquad (3)$$

Only one of the two equations (2) and (3) can hold good at a time for any feasible solution.

The objective functions, enumerated earlier, can be formulated as following :

(a) Minimising the make span : Add the following constraint

$$S_{i1(i,n)} + D_{i2(i,n)} \leq S_q \quad \text{for all } i \qquad (4)$$

where S_q is the make span time. The objective function is Min $Z = S_q$

(b) Minimising the idle time on machines : Add the constraints of the following type :

$$S_k \geq S_{ik} + D_{ik} \quad \text{for all } i \text{ and } k \qquad (5)$$

where S_k is the time of completion of all processes on machine k. The objective function is $Z = \min \sum_k S_k$.

(c) We can further extend the objective function by adding the following constraints :

$$SJ_i \geq S_{il(1,n)} + D_{il(i,n)} \quad \text{for all } i \qquad (6)$$

$$SM_k \geq S_{ik} + D_{ik} \quad \text{for all } i \text{ and } k \qquad (7)$$

where SJ_i, SM_k denote the time of finishing the last operation respectively on job i and machine k. If CJ_i and CM_k denote cost of per unit idle time respectively on job i and machine k, then the objective function can

be written down as follows :

$$\text{Min Cost} = \sum_{i=1}^{m} CJ_i * SJ_i - \sum_{k=1}^{n} \sum_{i=1}^{m} CJ_i * D_{il}(i,k)$$

$$\sum_{k=1}^{n} CM_k * SM_k - \sum_{k=1}^{n} \sum_{i=1}^{m} CM_k * D_{ik}$$

or, $$\text{Min } Z = \sum_{i} CJ_i * SJ_i + \sum_{k} CM_k * SM_k \qquad (8)$$

The constraints (1) and (4), or (5), (6) and (7) with appropriate objective function form a linear programming problem.

3.2 GREENBERG'S FORMULATION USING THE BRANCH AND BOUND TECHNIQUE

The equations (2) and (3) in 4.1 are combined by Greenberg as follows :

$$S_{ik} + X_{ij}^{k} * D_{ik} \leq S_{jk} + (1 - X_{ij}^{k})M \qquad (9)$$

where X_{ij}^{k} = 1 if job i < job j

= 0 if job j < job i (10)

M = arbitrarily large number.

If job i < job j then equation (9) becomes

$$S_{ik} + D_{ik} \leq S_{jk}$$

If job i > job j then,

$$S_{ik} \leq S_{jk} + M$$

Since M is an arbitrary large number, both equations (2) and (3) can be deduced from equation (9). Addition of equation (9) to the LPP mentioned in previous article,

makes it integer LPP. Let

$$V = S_{ik} - S_{jk} \leq X^{k}_{ji} M - (1 - X^{k}_{ji}) D_{ik}$$

Subject to integer restrictions on X^{k}_{ji}, V cannot have an arbitrary value. In other words for V to be non-restrictive there should be no integer restrictions on X^{k}_{ji}. A non-restrictive value of V implies no restriction is being imposed on S_{ik} and S_{jk} which is equivalent to saying that there are more than one machine of kind k. If this be the case the constraints (2) and (3) need not be specified. Specifying equation (2) or (3) puts restriction on S_{ik} and S_{jk} and thereby puts integer restrictions on X^{k}_{ji}. Hence integerization of X-variable is equivalent to adding to the LPP mentioned in previous article the restrictions of type (2) or (3). Therefore we need solve a non-integer LPP only.

There are $(n!)^m$ such X-variables which have to be assigned a 0 or 1. The constraints (1) and (4), (5), (6) or (7) being linear and convex the region of solutions is a convex polyhedron in a multidimensional space. Therefore, the branch and bound procedure can be applied for obtaining an optimal assignment of X-variables (see the reference 11). This procedure tells that the minimum value of the cost function Z, obtained by assigning a X-variable a 0 or 1, is the lower bound for any further assignment of yet unassigned X-variables. Therefore an assignment tree can be constructed based on branching from the lower bounds onwards. The nodes in this tree contain the value of the cost function Z for the partial assignment

$$\begin{matrix} 1 & n \end{matrix}$$

and the branches represent the assignment of an X-variable or equivalently the addition of the constraints of type (2) or (3). Those nodes which have not been chosen for downward branching are unmarked. The node with the lowest Z among the unmarked node is chosen for downward branching, in the tree. Some branches may lead to a node with no-feasible solution. Such nodes are never considered for downward branching. These nodes indicate that a looping sequence has been tried upon.

Proceeding in the above manner will lead to a node where all the X-variables would have been assigned and it has the least value of Z in comparison to all other unmarked nodes in the assignment tree. This is the optimal solution.

Selection procedure of node for downward branching may choose a node higher up the tree upsetting a number of assignments. This is the problem of backtrack and is a very serious problem. This has been considered in article 3.4. In the next articles we will look at the solution of the linear programming problem for calculating the value of any node in the assignment tree.

3.3 THE LINEAR PROGRAMMING PROBLEM

Greenberg has used simplex method to solve the LPP. However because of inherent simplicities in the constraints the method of obtaining an optimal solution turns out to be a simple procedure called the active feasible scheduling

procedure. This procedure has been used in all the heuristic methods of solving the job-shop problem used hitherto. The schedule thus obtained is called a Gnatt Chart. In this article an equivalence will be established between the active scheduling procedure and the Dual Simplex procedure.

Any complete solution to a job-shop problem is characterised by a particular assignment of integer valued X-variables, or in other words a particular sequence of jobs on all machines. Therefore an active feasible schedule also has such a particular sequence. Subject to such a sequence, an active feasible schedule is constructed as follows :

A job is assigned to a machine according to the specified sequences, at the earliest time at which both the machine and the job are free.

Such a procedure will generate a schedule in which no machine is idle for a length of time sufficient to process completely a schedulable job.

Let us now examine the LPP. Rewriting the LPP corresponding to the root node of the assignment tree,

$$\text{Min } Z = \sum_{i=1}^{m} CJ_i \cdot SJ_i + \sum_{k=1}^{n} CM_k \cdot SM_k$$

or $$\text{Max } Z = -\sum_{i} CJ_i \cdot SJ_i - \sum_{k} CM_k \cdot SM_k$$

subject to, $S_{il(i,\,k\,l)} - S_{il(i,k)} - D_{il(i,k)} \geq 0$

$$SJ_i - S_{il(i,n)} - D_{il(i,n)} \geq 0$$

$$SM_k - S_{ik} - D_{ik} \geq 0 \qquad (11)$$

$$S_{ij} \geq 0 \quad \text{for all i and j}$$

The coefficients matrix of the LPP contains elements, either 1, 0 or -1. There are a large number of zeros in this matrix. Storing and manipulation of this matrix is the essential part of simplex or Dual simplex procedure. This is enormous effort even for a small problem. In (11) the size of the coefficients matrix A_{pq} will be like,

$$p = m(n-1) + m + mn = 2mn$$

and $q = mn + m + n.$

The surplus variables and integerization equations of type (2) or (3) will increase this size further.

Further examination of the constraints of the LPP as given in (11) reveals that starting time of any operation on a job is dependent only upon the finish time of the operation which directly precedes it. Introduction of constraints of type (2) or (3) introduces the interjob precedence relations on a machine. This shows that there is an obvious flow like structure of the problem. Active scheduling procedure exploits this property.

In the Dual simplex procedure for a general form of LLP,

$$\text{Max } Z = c\,x$$

subject to $$A\,x \geq b$$

the necessary and sufficient conditions for obtaining an optimal solution are as follows :

$$X_{br} \geq 0 \quad \text{for basic variables} \tag{12}$$

$$Z_i - C_j = C_{Bi}\, Y_{ij} - C_j \geq 0 \quad \text{for all variables} \tag{13}$$

where C_{B_i} : cost of ith basic variables

C_j : cost of jth non-basic or basic variable

and (Y_{ij}) : the coefficient matrix in the tableu at any iteration.

In the LPP for the job-shop problem all these costs are negative.

To establish the equivalence in the Dual simplex and the active scheduling procedure, following theorem can be stated.

Theorem 1: For the given set of constraints the active scheduling is an optimal schedule, i.e. it satisfies equations (12) and (13).

Proof

In an active schedule a job is assigned to a machine at the earliest time at which both machine and job are free. Therefore all the variables in the constraints are greater

or equal to zero. So the active schedule is a feasible schedule. This satisfies the criterion (12).

The price variables SJ_i (i = 1,2...m) and SM_k (k=1,2..n) and all the start-time variables except a few $S_{il(i,1)}$ have positive nonzero values so they will be in the set of basic variables. The nonzero time gaps between any two operations on a job or a machine will also be included in the basic set. These are the surplus variables in the Dual simplex formulation.

Any row in the optimal tableu can be written as follows :

$$S_i + \sum_{\text{all } j} Y_{ij}(X_{pq})_j = b_i$$

where

S_i : basic variables

b_i : value of the basic variables

$(X_{pq})_j$: those $S_{il(i,1)}$ for any i or surplus variables which have zero values, i.e., the nonbasic variables.

For the active schedules we can also write similar equations for the nonzero variables S_i in terms of zero valued variables $(X_{pq})_j'$.

$$S_i + \sum_{\text{all } j} Y_{ij}' (X_{pq})_j' = b_i' \qquad (14)$$

This will be a tableu similar to the form obtained in the

Increasing $(X_{pq})_j$ above zero level is equivalent to pushing a job to the right of the minimum time when both the machine and job are free. Such an assignment can push some or all the basic variables to have more positive value than what they will have if all the assignments are done at the earliest possible times. This corresponds to a situation where in equation (14) the Y_{ij} are either 0 or -1.

Then the optimality criterion (13),

$Z_j - C_i = C_{B_i} \cdot Y_{ij} - C_j \geq 0$ for nonbasic j

because all the costs are negative and Y_{ij} are either 0 or -1. Since both the criteria (12) and (13) are satisfied by the active schedule, it is an optimal schedule. Note that it is optimal with respect to the constraints given in the corresponding LPP.

QED.

Addition of a new integerizing equation of the type (2) or (3) to a LPP may make the present solution to be nonfeasible. The additional equation can be added to the tableu of the dual simplex procedure after expressing all the basic variables in it by the nonbasic variables of the original LPP and including the surplus variable into the basic set.

If the additional equation is

$$S_{ik} \quad S_{jk} \quad D_{jk} \qquad (14)$$

introducing the surplus variable x, it becomes

$$S_{ik} - S_{jk} - x = D_{jk} \qquad (14)$$

If S_{ik} and S_{jk} are the basic variables in the original LPP, then using the equations similar to (13), they can be expressed in terms of the nonbasic variables. Then surplus variable x will be included in the basic set. Since nonbasic variables do not have any price the $Z_j - C_j$ 0 remains intact.

If job i and job j are not in clash over the machine k as determined by the solution to the original LPP, then x will have positive value otherwise negative. Hence the solution to the original LPP together with the new basic variable x is a non-feasible with all $Z_j - C_j \geq 0$. Therefore it can be used as initial solution for the iterations on the new LPP. The post optimization over this initial solution will require a few iterations.

The active schedule for the new LPP will be constructed as follows :

(1) Add the additional constraint on sequence to the assignment tree. Mark all the operations as "unassigned".

(2) Search the tree to find a leftmost operation on a job which preceeds all other remaining unassigned operations on all the jobs. Assign it to the appropriate machine at the earliest time at which both the machine and the job are free.

(3) Mark this operation as "assigned". If all operations are assigned the schedule is complete, otherwise go to the step (2).

This procedure will cost more computational time than the post optimization in the dual simplex procedure. However in the branch and bound procedure any backtracking will require construction of the new tableu by the search of the appropriate portion of the assignment tree because the storing of tableu corresponding to each node is a huge effort. It will require considerable manipulation than forming an active schedule. Another point in favour of the active schedule is that it need store only the assignment tree while the Dual Simplex procedure will have to store the coefficient matrix or the Tableu in addition to the assignment tree.

Storing a large matrix is a difficult problem on the presently available computers. Using backup stores like disks or tapes will involve mechanical movements for any manipulation in the matrix which will be very costly. Therefore the choice rests with the active scheduling procedure.

A further reduction in the calculations of active scheduling, i.e. preparing Gnatt chart, is possible by modifying the branch and bound procedure used by Greenberg. It will be dealt in a later article.

3.4 LIMITATIONS OF THE GREENBERG'S METHOD

At any stage of scheduling, Greenberg fixes an integer variable to be zero or one, which is equivalent to fixing the relative sequence between two jobs on a machine. If there are m jobs, then each job requires fixation of (m-1) sequences before its position in the schedule is fixed. However, the starting-time of an operation can be fixed only when the starting times of the operations preceding it are fixed in advance. As we have seen in the last article in the construction of the active schedule or the Gnatt-chart, we have to search the tree each time we fix the start time of an operation. So the construction of the Gnatt chart-becomes very time consuming.

In the branch and bound procedure one has to back-track to all those 'unmarked' (or unbranched) nodes which have a cost lower than the cost of the optimal schedule. The back-tracking can be minimised by fixing those sequences earlier which give large difference between the lower and the upper bounds. Greenberg fixes the sequences in a completely arbitrary manner, and also those sequences are fixed in which the jobs concerned have no clash. Besides, fixing of one sequence can be expected to cause only a small increase in the cost or the objective function. Also the difference between the lower and the upper

bounds will not be sufficient to stop the backtracking. Moreover we do not expect that a job will be sequenced earlier to a job which is placed much earlier than it in the schedule. Therefore, fixing such a sequences is just a futile effort. So the problem of backtracking is really big.

Another limitation of Greenberg's method lies in the calculation of the costs of the partial schedule, or the bounds. According to his method the bounds are calculated from the follow equation

$$Z = \sum_{i=1}^{m} CJ_i \cdot SJ_i + \sum_{k=1}^{n} CM_k \cdot SM_k \qquad (15)$$

where SJ_i = time of finishing the last operation on job i and SM_k = time of finishing the last operaion on machine k. These times are based on the schedule with partial fixation of sequences. If all the jobs on a machine are in clash with each other than the SM_k will be even less than the sum of the processing times of all jobs on that machine.

All these limitations tell us that several modifications are needed in the Greenberg's branch and bound procedure. The modifications are suggested in the following articles serially.

3.5 FIXING OF MORE THAN ONE SEQUENCE AT A TIME

It is necessary to fix more than one sequence at a time the lower bound calculated should be nearer to the probable optimal cost, to reduce the backtracking. Two methods can be suggested as follows :

1) Fix the sequence of an unassigned job with respect to all the other unassigned jobs requiring processing on the same machine. If there are r such jobs then r nodes must be calculated, each node specifying the job that proceeds all others. Since one of these will definitely be assigned the first place among these jobs in the probable optimal schedule, therefore, it will give us the lower bound.

It is equivalent to assigning a value 1 to r integer variables as against to only one in Greenberg's method. It would have taken calculation of 2r nodes plus the necessary backtracking to fix an operation like the above. The price is paid in terms of the fact that if backtracking becomes necessary a greater amount of calculation is needed. In greenberg's procedure backtracking to an earlier stage changes only a few sequence, while in this method all the r sequence, while in this method all the r sequences are upset. This price is not much because the difference between the lower bound and the upper bounds in this method is larger than those in Greenberg's method.

The improvements of this method over the Greenberg's procedure will be further elaborated after presenting the second modification.

2) Consider the immediately next unassigned operations on all jobs. Calculate the nodes correspondings to fixing each such operation earlier to any other operation requiring some machine. If there are m such jobs then m nodes will be calculated. One of these operations will definitely be placed at its decided position, therefore it will give the lower bound of the schedule.

If an operation on a machine **k** is chosen to be assigned prior to other(r-1) unassigned operations on the same machine, then the above process is equivalent to fixing **r** integer values to one. Hence as regards its comparison with Greenberg's method the arguments given in the first method also hold here.

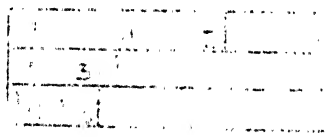
As we can see there is an automatic selection of next clashes to be resolved in the above method. In the first method there is a problem of finding a suitable machine on which the clashes must be resolved. This can be solved by choosing a machine which requires processing by atleast one of the immediately next unassigned operations on all thejobs. Such a set of operations can be called a MIX.

With this we will compare the two methods by the help of an example. Consider a following problem

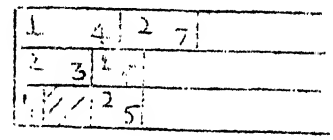
Job 1	1(4)	2(3)	The costs of job lateness
Job 2	2(3)	1(2)	and machine idle time are
Job 3	1(1)	2(2)	unity.

The numbers inside brackets are the processing times and outside the brackets are the machine number. The chart in the first procedure is as follows :

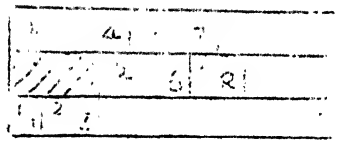
(1)



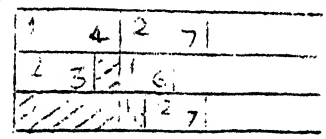
(2)



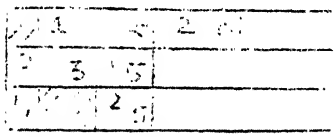
(3)



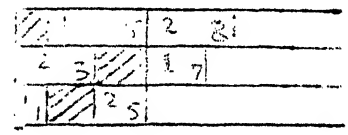
(4)



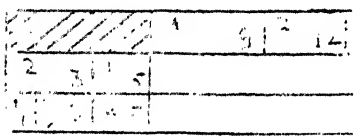
(5)



(6)



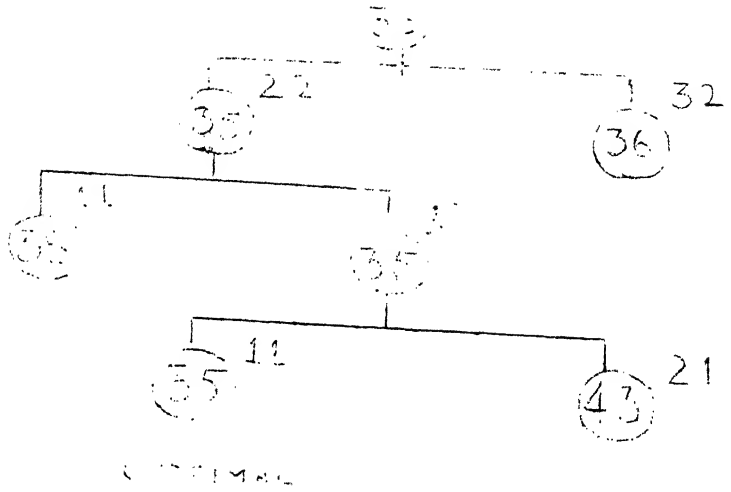
(7)



The superscript is the machine number.

The subscript is the actual time of the completing the operation.

Fig.1



The assignment Tree

Fig.2

The notations observed in the solution are as follows :

Operation **ik** : Operation of ith job on the **k**th machine.

Δ^1 : minimum incremental cost of job lateness in the schedule due to machine 1.

Δ^2 : minimum incremental cost of job lateness in the schedule due to machine 2.

$\Delta_1^{\mathbf{k}}$: incremental cost of job lateness because of 1th sequence on the **k**th machine.

Δ : largest of the minimum incremental costs of job lateness.

'ik$\ll$ jk' : Operation ik directly preceeds jk.

For the calculation of the root node, see the Gnatt chart No.(1) in the Fig.1.

$SM_1 = 7$ $\qquad$ $SJ_1 = 7$

$SM_1 = 8$ $\qquad$ $SJ_2 = 5$

$SJ_3 = 3$

Machine 1 : It is obvious that the operation 31 should not be placed as the first job on machine 1.

sequences : $11 \ll 31 \ll 21$; $\Delta_1^1 = 6$

$11 \ll 21 \ll 31$; $\Delta_2^1 = 7$

$31 \ll 11 \ll 21$; $\Delta_3^1 = 3$

$31 \ll 21 \ll 11$; $\Delta_4^1 = 5$

$$\Delta^1 = \min (\Delta_1^1, \Delta_2^1, \Delta_3^1, \Delta_4^1) = 3$$

Machine 2 : It is obvious that operation 12 should not be placed as the first job on the machine 2.

sequences : $22 \ll 32 \ll 12$; $\Delta_1^2 = 3$

$32 \ll 22 \ll 12$; $\Delta_2^2 = 5$

$22 \ll 12 \ll 32$; $\Delta_3^2 = 6$

$32 \ll 12 \ll 22$; $\Delta_4^2 = 7$

$\Delta^2 = 3$

Since we have not considered how does a sequence on one machine affects the sequence on the other machine, we

cannot add the Δ^1 and Δ^2. However, the largest of Δ^1 and Δ^2 will be the minimum increase in the cost function Z. Hence $\Delta = \max(\Delta^1, \Delta^2) = 3$

$$Z = SM_1 + SM_2 + SJ_1 + SJ_2 + SJ_3 + \Delta$$
$$= 7 + 8 + 7 + 5 + 3 + 3 = 33.$$

The MIX in Gnatt chart (1) has the operations 11, 22 and 31. Both machines 1 and 2 are involved in this mix. The operation 31 gives a push of 1 unit to other operations on the machine 1, which is also the least push given by any other operation on machine 1, in the mix. The minimum push on machine 2, as given by operation 22 is 2 units. Let us then choose the machine 2 for resolving the clash for the first placing on it, because by a larger push it is more likely to ease clashes among other operations.

On the machine 2 only jobs 2 and 3 are the contenders for the first place.

Operation 22 in first place

See the Gnatt chart No.(2).

$SM_1 = 7$ $\quad SJ_1 = 7$

$SM_2 = 8$ $\quad SJ_2 = 5$

$SJ_3 = 5$

Machine 1 : $11 \ll 31 \ll 21$; $\Delta_1^1 = 6$

$11 \ll 21 \ll 31$; $\Delta_2^1 = 7$

$31 \ll 11 \ll 21$; $\Delta_3^1 = 3$

$31 \ll 21 \ll 11$; $\Delta_4^1 = 5$

Machine 2 : $22 \ll 32 \ll 12$; $\Delta_1^2 = 1$

$22 \ll 12 \ll 32$; $\Delta_2^2 = 4$

$\Delta^2 = 1$

$\Delta = 3$

$Z = 7 + 8 + 7 + 5 + 5 + 3 = 35$

Operation 32 in first place

See the Gnatt chart No.(3).

$SM_1 = 8$ $SJ_1 = 7$ $SJ_2 = 8$

$SM_2 = 8$ $SJ_3 = 3$

Machine 1 : $11 \ll 31 \ll 21$; $\Delta_1^1 = 4$

$31 \ll 11 \ll 21$; $\Delta_2^1 = 1$

$\Delta^1 = 1$

Machine 2 : $32 \ll 22 \ll 12$; $\Delta_1^2 = 2$

$32 \ll 12 \ll 22$; $\Delta_2^2 = 4$

$\Delta^2 = 2$

$\Delta = 2$

$Z = 36$

The operation 22 has the lower bound so it will be placed as the first job on machine 2.

The new MIX : 11, 21 and 31.

Machine Chosen : Machine 1 (only one machine in the mix)

The operation contending for the first place on machine 1 are 11 and 31.

Operation 11 : first job on machine 1

See the Gantt chart No.(4).

$SM_1 = 7$ $SJ_1 = 7$ $SJ_2 = 6$

$SM_2 = 9$ $SJ_3 = 7$

Machine 1 : $11 \ll 21 \ll 31$; $\Delta_1^1 = 2$

$11 \ll 31 \ll 21$; $\Delta_2^1 = 1$

$\Delta^1 = 1$

Machine 2 : $22 \ll 12 \ll 32$; $\Delta_1^2 = 2$

$22 \ll 32 \ll 12$; $\Delta_2^2 = 3$

$\Delta^2 = 2$

$\Delta = 2$, $Z = 38$.

Operation 21 as the first job on machine 1

See the Gantt chart No.(5).

$SM_1 = 7$ $SJ_1 = 8$ $SJ_2 = 5$

$SM_2 = 8$ $SJ_3 = 5$

Machine 1 : $31 \ll 11 \ll 21$, $\Delta_1^1 = 3$

$31 \ll 21 \ll 11$; $\Delta_2^1 = 4$

$\Delta^1 = 2$

Machine 2 : $22 \ll 32 \ll 12$; $\Delta^2 = 0$

$\Delta = 2$, $Z = 35$.

The operation 21 has the lower bound so it will be placed as the first job on machine 1.

In the Gnatt chart corresponding to operation 31 we observe that 32 is automatically schedule as it is the next operation after 31, and it has no clash with the remaining operation on the second machine, i.e., the operation 12.

The new mix : 11 and 21

The machine chosen : Machine 1

The contenders for 2nd place on Machine 1 are 11 and 21.

Operation 11 in the second place

See the Gnatt chart No.(6).

$SM_1 = 7$ $SJ_1 = 8$ $SJ_2 = 7$

$SM_2 = 8$ $SJ_3 = 5$

Machine 1 : No clashes, $\Delta^1 = 0$

Machine 2 : No clashes, $\Delta^2 = 0$

$Z = 35$

Operation 21 in the second place

See the Gnatt chart No. (7).

$SM_1 = 9$ $SJ_1 = 12$ $SJ_2 = 5$

$SM_2 = 12$ $SJ_3 = 5$

Machine 1 and 2 : No clashes

$Z = 43$

The operation 11 is scheduled in the second place on the machine 1. With this all the clashes are resolved so the scheduling stops. Note that Z is not the actual cost. The cost is calculated as the sum of the job-idle times and machine idle-times multiplied by their cost per unit time. The Optimal cost is 5. The assignment tree is given in Fig.2.

This problem has been solved by H.Greenberg (10) to minimise only the idle-times on the machines in 19 nodes after solving 19 linear programming problems.

The calculation of bounds for a general mxn shop problem are discussed in greater detail in article 6 of this chapter.

Fortunately, no backtracking is required in this problem. If backtracking becomes necessary then a search of the tree becomes necessary to find the node from where the further branching will be resumed. The Gnatt chart will be constructed according to the sequences fixed while traversing the path from the root node to the chosen node.

The assignment tree for the same problem using the second method is given in Fig.3. The bounds are calculated in the same way as that in the first method.

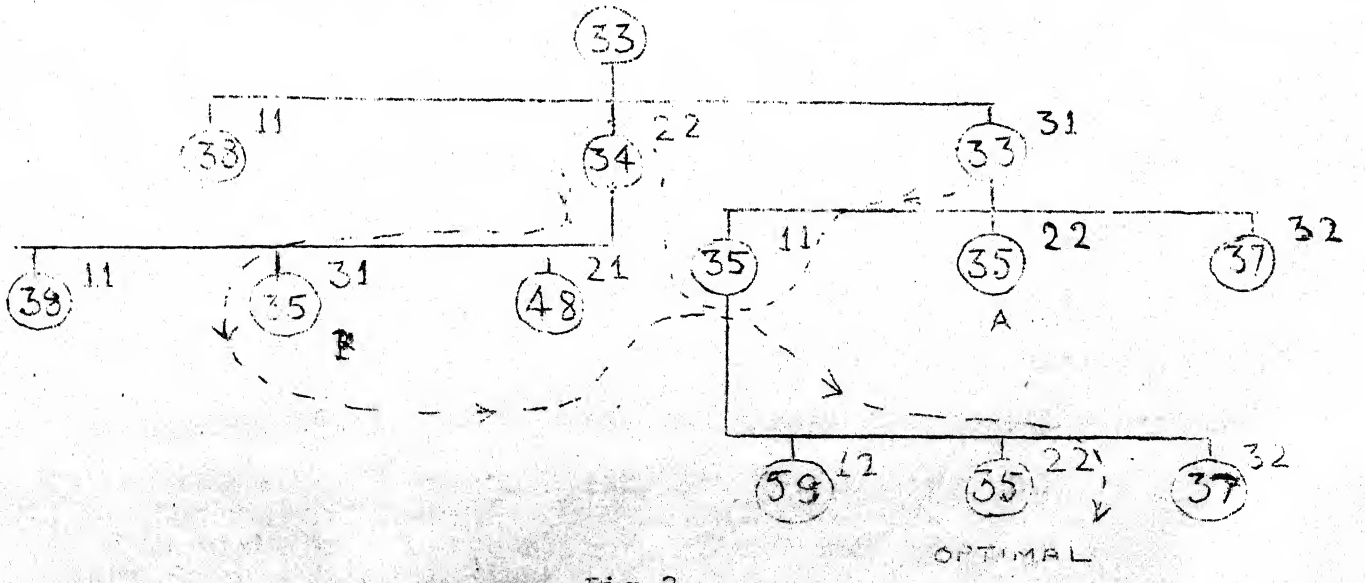


Fig.3

The dotted path shows the branch and bound path adopted by the scheduler. In this method we calculate three nodes at each stage of the tree while, in the first method this is not a fixed number; it can be at most 3 equal to the number of jobs. Note that in the above tree nodes A and B are identical yet they require to be duplicated. This situation arises because at the first stage of tree we have compared nodes marked 22 and 31, i.e., those operations which require operation on different machines. There is a definite possibility of both being scheduled together.

This prompts us to think of scheduling more than one operation out of the mix of the immediately next unassigned operations on all jobs, the one we talked about earlier. This process will be equivalent to simultaneously attending the queues on different machines which occur in the mix. It can be easily argued that we can be completely sure that fixing one operation in the mix will give a lower bound but not so about fixing more than one operation in the mix even though they require processing on different machines. For that then we have to consider the combinations amongst all those operations, not necessarily in the mix, on various machines that are in the mix, and are in clash with each other. This may swell to a large number. The number of nodes

corresponding to all the combinations of them will be even larger. So it is better to leave it.

Let us come back to the comparison between the first and the second method. It is quite possible that the operation corresponding to the lowest bound in the first method may turn out to be one whose predecessor operation is still in clash with other operations, that is, it is not in the mix. If the predecessor is shifted to the right as a result of some clash resolution then the earlier calculation may be upset, i.e., the back-tracking may become necessary. Besides this, the Gnatt chart construction becomes difficult. In the second method each node can be labelled with the starting time of the operation concerned. In the first method we will have to search the tree to find out how many operations to the right of the recently scheduled operations get pushed to the right. However, there is a way out if we utilize the idea of a Mix. If any operation corresponding to the lowest bound is not a member of the Mix, then keep this machine aside. Choose another machine from the mix and again find the lowest node and repeat the above test. We will definitely find one operation, as it is assured in the second method. This will save us for complex Gnatt chart construction for each node. The price is paid in terms of the calculation of a larger number of nodes. However, it may happen that we do not have to

consider any new node. If all the operations in a mix require operation on a single machine then no extra calculations are needed. If there are more than one machine in the mix, then there is a possibility of simultaneous scheduling of one operation on each of these machines. Therefore, it is a safer bet against large computer time in constructing Gnatt charts for all the nodes in the tree. Moreover, with the new arrangement in the first method, we will be able to select a clash which gives large shifting of the schedule to the right. This is likely to ease clashes among the remaining operations. This is not possible in the second method, because the choice of the lowest bound may fall upon an operation which gives least increase in the cost. The least increase in the cost will usually produce small right shifts in the schedule.

From the point of view of the number of nodes to be calculated at any stage in the tree and the duplication of nodes, the first method weighs higher than the second one. So from now on we will restrict our attention to the first method. For ease of reference we can refer it as the alternate procedure in further discussion.

In the Gnatt chart no.(1) of Fig.1 we have seen that for deciding the first place on machine 2 we considered only jobs 2 and 3. Out of all the three jobs,

the earliest time of finishing is for the job 3. Both jobs 2 and 3 have the start times less than this time. The operation 12 on job 1 has a starting time of 4, which is larger than the earliest finishing time. So we know that as far as the first placing on machine 2 is concerned, the job 1 is not a contender. This observation can be generalised as follows :

For deciding any placing on a machine, only those unassigned operations must be considered which have their start-time less than the time of finishing an unassigned operation which has the least time of finishing in the present Gnatt chart.

This will reduce the number of node calculations and the size of the assignment tree. Therefore, the number of the backtracking steps can also be expected to decrease.

We are now in a position to put down the alternate procedure in the form of an algorithm as follows :

(1) Form the mix of "leftmost unassigned operations on all jobs. If any operation in the mix is not in clash, then declare it 'assigned ' and replace its position in the mix by its successor operation.

(2) Choose a machine out of this mix, operations on which produce the largest of the minimum right shifts in other operations in the Gnatt chart.

(3) Calculate the bounds corresponding to those operations on this machine which are in 'clash', as discussed earlier. If the lowest of these bound is corresponding

to an operation which is not in the mix, then choose another machine according to step(2) and calculate the new nodes. Repeat this process till a lowest bound is found corresponding to the operation in the mix.

(4) Search the lower bound among the newly calculated nodes and all the unbranched nodes in the tree. Further branching is done from this node onwards. Update the tree.

A backtracking step at this stage may lead to a node the associated operation of which is ahead of its corresponding mix. In such a case go to step (3) where the new clashes are resolved on a machine that requires processing by the operation which is preceding the currently assigned operation and is in the current mix. In subsequent branchings Gnatt-charts are constructed such that no sequences are violated that have already been fixed in the current branch. This procedure is followed only till there remains no assigned operation which is ahead of mix.

If the above change is not required go to step (1). The procedure is stopped when all the clashes are resolved.

Next we shall compare the alternate method and the modified Greenberg's procedure. The idea of mix can also be introduced in the Greenberg's method. The modified procedure is as follows :

(1) From a mix of those leftmost unassigned operations on each job. The unassigned operations are those whose sequences with all other operations

on the same machine are not resolved completely. If such an unassigned operation does not have any clashes then it is removed from the mix and its successor is placed in its position.

(2) Choose a machine in the mix in the same manner as in the alternate procedure. On this machine choose two operations which produces the largest minimum shifts relative to each other. Calculate the nodes corresponding to two sequences for these operations and calculate the bounds. The sequence with the lower bound is selected.

(3) Update the mix and the tree. If all clashes in the schedule are not resolved go to step (2), otherwise stop.

Let us then solve the 3x2 problem, solved earlier by the alternate procedure.

The root node is same as that calculated in the alternate procedure.

The machine chosen from the mix is likewise 2.

The operations in clash over this machine are 22 and 32.

Following are thecalculations of those nodes which were not calculated in the alternate procedure. Rest of the calculations are left to avoid repetition.

Sequence 22 < 32: See Gnatt chart No.(2) in Fig.(1).

$Z = 35$

Sequence 32 < 22: See Gnatt chart No.(3) in Fig.(1).

$Z = 36$

The node marked 22 < 32 is selected.

New mix : 11, 21, 31

Machine Chosen : Machine 1

Operations Chosen : 11 and 31

Sequence 11 < 31: See Gnatt chart No.(1) in Fig.4

$SM_1 = 7$ $SJ_1 = 7$ $SJ_2 = 5$

$SM_2 = 9$ $SJ_3 = 7$

Machine 1 : 11 ≪ 31 ≪ 21 ; $\Delta_1^1 = 2$

11 ≪ 21 ≪ 31 ; $\Delta_2^1 = 1$

21 ≪ 11 ≪ 31 ; $\Delta_3^1 = 10$

$\Delta^1 = 1$

Machine 2 : 22 ≪ 32 ≪ 12 ; $\Delta_1^2 = 3$

22 ≪ 12 ≪ 32 ; $\Delta_2^2 = 2$

$\Delta^2 = 2$ $Z = 37$

Sequence 31 11 : See Gnatt chart No.(4) in Fig.1. $Z = 35$.

The node marked 31 < 11 is selected.

In the Gnatt chart corresponding to this node, the operations 31, 32 and 22 have no clashes so they are declared "scheduled" as far as this branch of the tree is concerned.

The new mix : 11 and 21.

Sequence 11 < 21 : See Gnatt chart No.(6) in Fig.1.

$Z = 35.$

Sequence 21 < 11 : See Gnatt chart No.(7) in Fig.1.

$Z = 43.$

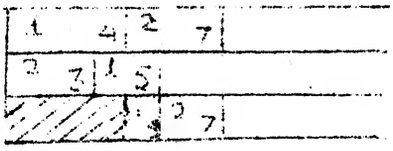


Fig.4

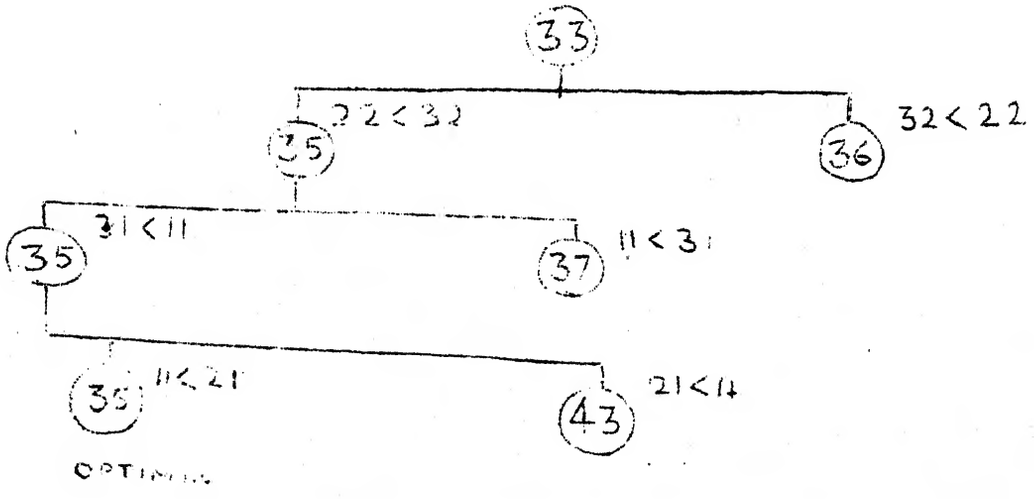


Fig.5

In considering the sequences on various machines for the calculation of we have to be very cautious not to violate the already resolved sequences on the tree. For this we must keep track of the resolved sequences on each operation individually. In the alternate procedure we need track only the "scheduled" operations which is relatively a smaller job.

This method also gives the solution in seven nodes. It so happens because the jobs fit in very well with each other in this problem.

In constructing the Gnatt chart in the modified Greenberg's procedure, one has to be cautious that no previously fixed sequences are violated. Therefore, every chart must be constructed from the beginning maintaining the sequences already fixed. In the alternate procedure we need only reform the previously obtained chart by pushing all those unassigned operations which are in clash with the selected operation. Only when a backtracking step is required the Gnatt chart is built from the beginning. Hence the construction of chart in the alternate procedure is very easy.

Let us compare for a general case the number of nodes which must be calculated to solve a problem. Suppose on certain machine in the mix there are r jobs in clash with each other. Note we shall consider only those jobs to be in clash which are in clash with an operation with the earliest completition time. The alternate method will require,

$$r + (r-1) + (r-2) + \ldots + 3 + 2 = \frac{r(r+1)}{2} - 1$$

number of nodes to determine the complete sequence of r jobs.

In the modified Greenberg's procedure, for the job which will be placed first, sequences with r-1 jobs must be resolved. Out of the remaining r-1 jobs, the next placing is decided in at most r-2 sequences. However if the resolving of earlier r-1 sequences remove clash among some of these r-1 operations then the sequences to be resolved next will be less than (r-2). The maximum number of sequences to be resolved is as follows ,

$$(r-1)+(r-2)+(r-3)+\ldots+3+2+1 = \frac{r(r-1)}{2}$$

The corresponding number of nodes is r(r-1).
The minimum number of nodes can be easily argued and it is 2(r-1). The number of actual nodes calculated depends on the combinations of the processing times, the costs of job-lateness and the selection of the operations and the machine for resolving the clashes.

A table for different values of r is given for comparison.

	r=2	r=3	r=4	r=5	r=6
$\frac{r(r+1)}{2}-1$	2	5	9	14	20
r(r-1)	2	6	12	20	30
2(r-1)	2	4	6	8	10

The alternate method has a number which is the exact arithmatical mean of the maximum and the minimum number

of nodes in the modified Greenberg's procedure. For alternate procedure this number is independent of the combinatorial structure of the problem.

These numbers are little deceptive because they strongly depend upon the backtracking. The backtracking in the alternate method will upset more sequences than in the modified Greenberg's procedure. However, because of fixing (in the alternate method) the sequences amongst more than one jobs all together increases the discriminatory power of the bounds and so the number of backtracking steps in this method can be expected to be smaller than in its counterpart.

The above comparison establishes that the alternate method is relatively less depadent on the combinatorial nature of the problem and the price paid for it is also only moderate. For the problems like job-shop we cannot define a closed form formula to prove the superiority of one procedure over another. We can compare them only in terms of the time taken for solving and the amount of dependence of the procedure onto the structure of the problem data.

3.6 THE ALTERNATE METHOD AND THE CRITICAL PATH METHOD

The scheduling of m jobs on n machines can be considered as a project. The critical path method (CPM) can be used to construct the tree of sequences for this

project. This tree is constructed in much the same manner as the alternate method does. The node represents the fixation of an operation on some job. The CPM enumerates all the branches leading from the start to the end. The path that gives the least cost is the optimal solution. It is essentially a complete enumeration procedure. The alternate method discards at every stage of the tree, those nodes which have bounds (costs) higher than the least bound among all the unbranched nodes in the tree. Thus the size of the tree in the alternate method will atmost be equal to that of the CPM but on an average it will be much smaller than the later. Increase in the number of jobs increases the size of the tree in CPM at a much faster rate than in the alternate method. Hence the alternate method will be much faster than the CPM.

3.7 THE BOUNDS

The bounds are the evaluations of the cost function Z resulting from the partial assignment of the sequences. The node with the least bound in the assignment tree is chosen for further branching downward.

Brown, Lomnicki, J.N.D. Gupta (15) and others while working on simpler job-shop problems have reported that the amount of backtracking depends upon how good the lower bounds are. However, the larger the difference between the lower and the upper bounds, the lesser is the probability

of reversing the choice. Therefore not only the lower bounds, but also the difference between the bounds is an important criteria to reduce the backtracking.

As mentioned in article 4 of this chapter, we must evaluate the SM_k, the time of completion of the last operation on the kth machine more cautiously. Brown, Lomnicki and J.N.D. Gupta have used for SM_k the time of finishing the last operation on kth machine among the so far scheduled operations plus the sum of the processing times of other remaining operations on this machine. They have not considered the clustering of these jobs. For example consider an incomplete schedule in which the start times and durations of unassigned operations on the kth machine are as follows :

	JOB 1	JOB 2	JOB 3	JOB 4
Starting time	0	1	8	10
Duration	4	1	3	5

Brown's bound will give,

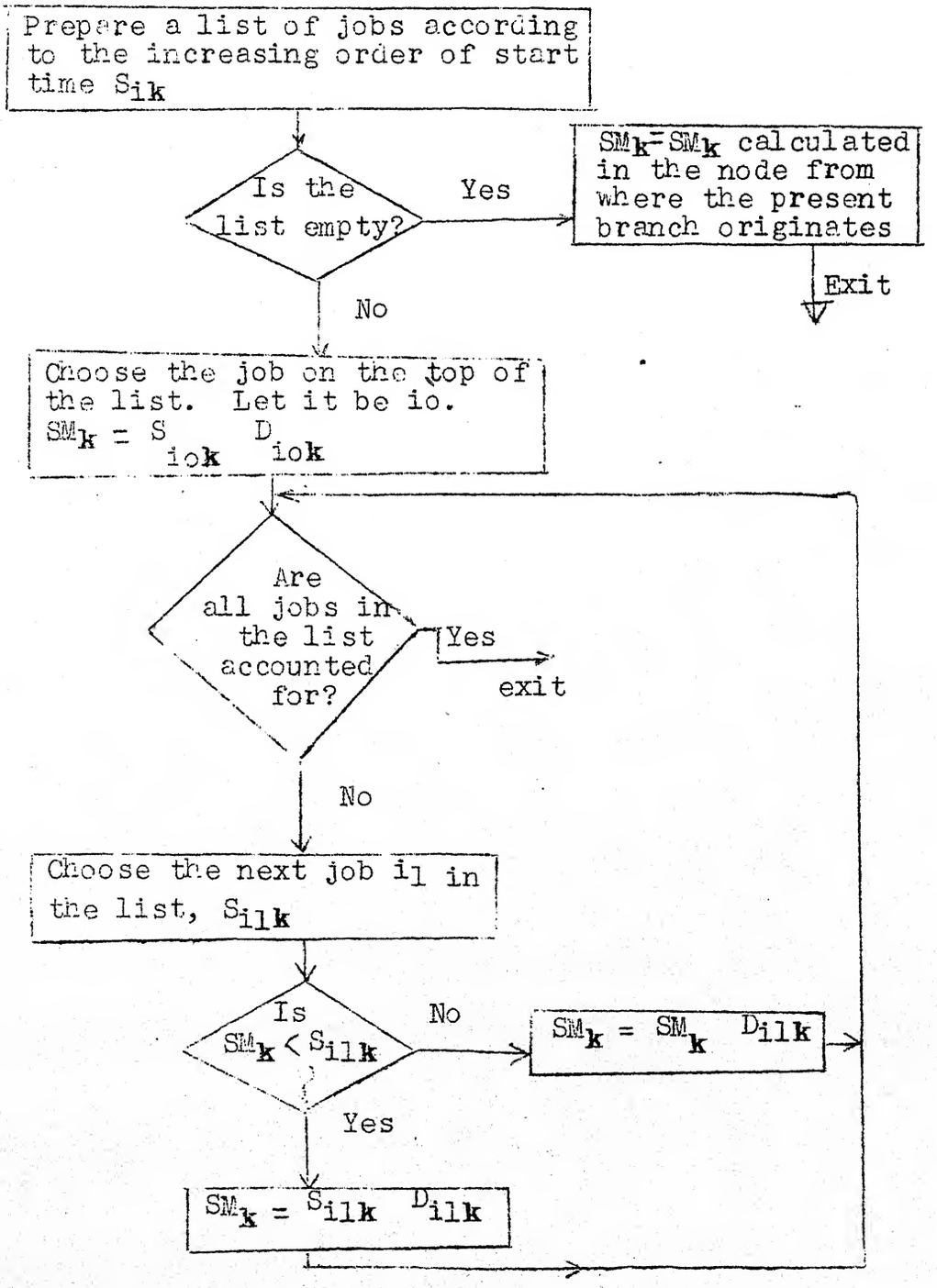
$$SM_k = 4\ 1\ 3\ 5 = 13.$$

If we draw a Gnatt chart and add to the above sum the unavoidable slack times, the SM_k comes out to be 16. This difference will be enhanced in a problem with large number of jobs and non-identical orders of processing of different jobs on the various machines.

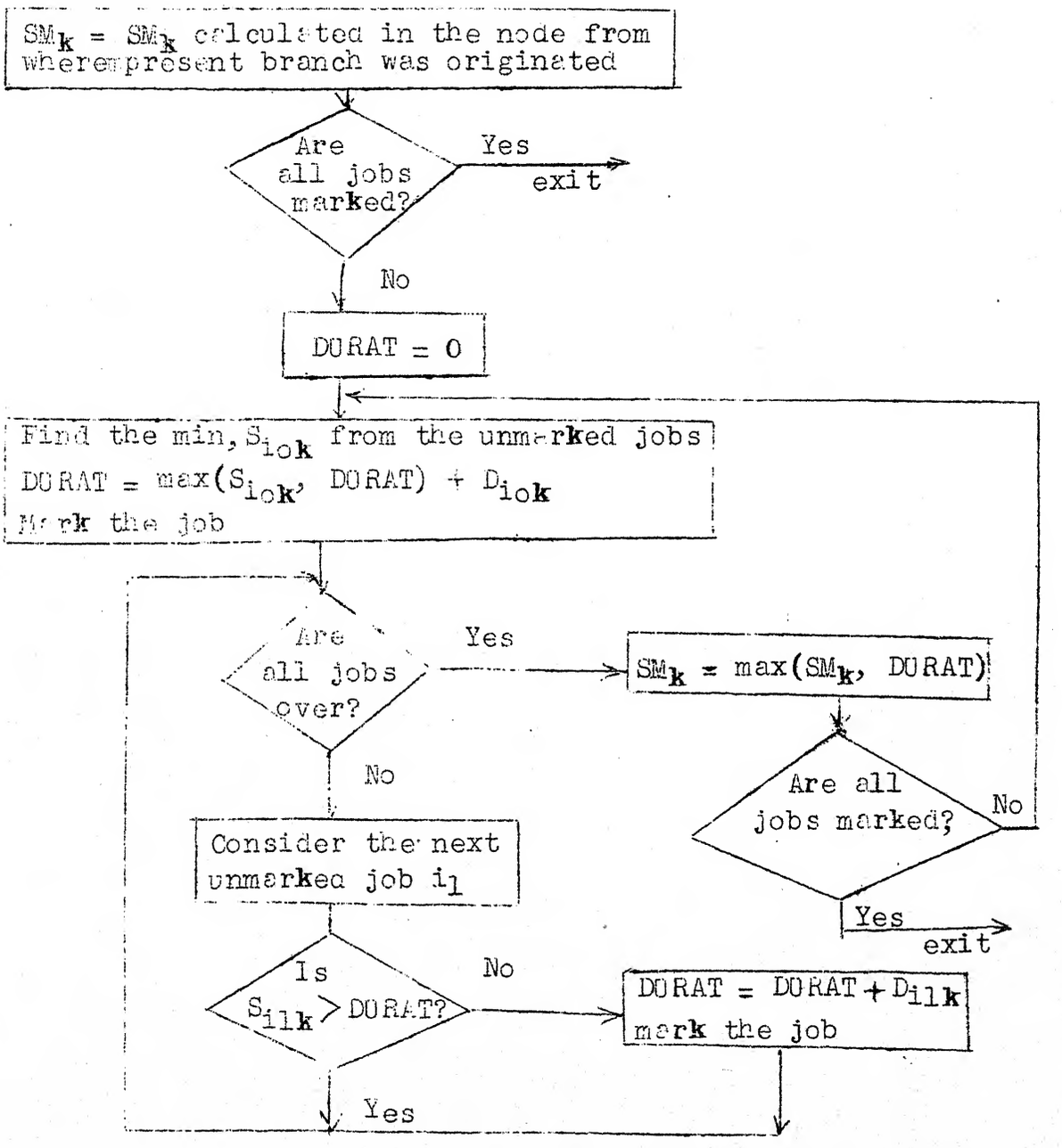
Since it is cumbersome to represent the calculation of SM_k by a mathematical expression, it will be presented by flowchart as follows :

All the jobs referred to in the following charts are the unassigned or partially assigned jobs, that is, those jobs whose sequences on a machine are not completely determined.

(1) Calculation of SM_k :



A faster chart is as follows :



In this flowchart we need not prepare the list according to increasing order of start times of the operations ik's. In preparing the list we will have to search as many minimum start times as the number of unassigned operations. In the above chart if all the operations on k are in clash with some one or other, then we have to find the minimum only once.

Next we will estimate the effect of SM_k on the schedule. If no operation on machine k has a finish-time equal to SM_k then all the unassigned operations must be considered for possible shifting to the SM_k or more accurately each of the unassigned operations must be tried as a last job on the kth machine. The resulting minimum increase in the cost can be added to the cost function Z. This is the approach J.N.D. Gupta and others have adopted in their methods to solve a less general scheduling problem.

Trying each operation as the last operation on machine k affects the job completion time of only one job. If there is clustering of unassigned operations in the left end of the schedule, then we can get a higher increase in the minimum cost by trying each operation to be scheduled earlier to others. This will shift more than one job towards right and thus affecting schedule in a wider sense. Let the minimum increase in cost be Δ^1.

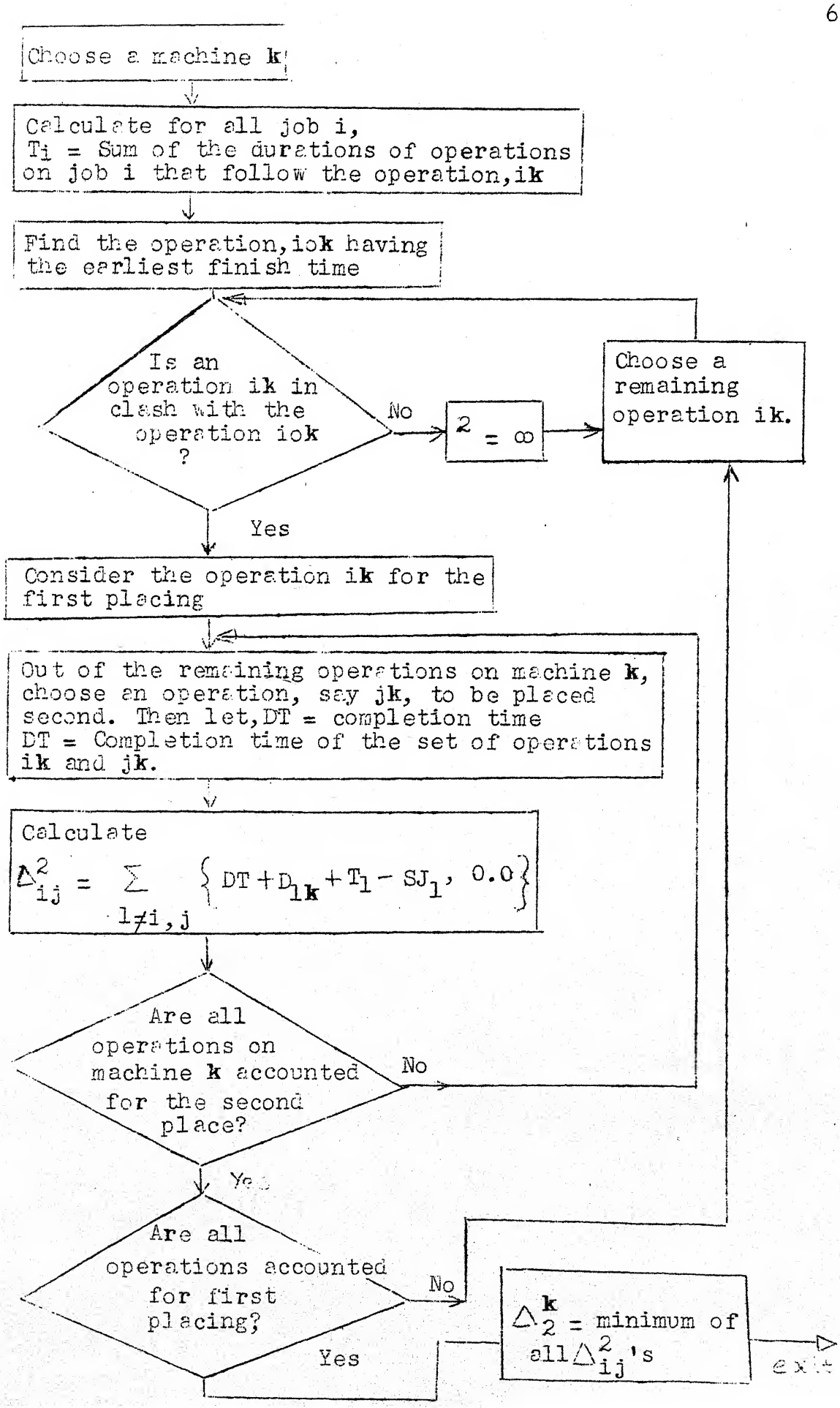
If an unassigned operation at the left end of the schedule does not clash with any other operation on the same machine then Δ^1 will be zero. In such a case we can try for fixing first and second placings of operations on this machine and decide the minimum increase in cost to be Δ^2. In this way we can decide the minimum increments due to third, fourth and so on placings. If at any stage there are r unassigned operations on a machine, then for

deciding the minimum increase in cost due to all possible placings, we will have to consider r! permutations. We would like to fix only the first and the second placings. The second placing has been included so that our calculations are not marred if the first placed operation has no clashes. It will require the searching of r(r-1) combinations.

The Δ^2's are calculated for each machine separately, and the maximum of them is chosen to be minimum increment Δ in the total cost. If we want to consider simultaneous effect of SM_k's on the job late-times then we will again have to consider r(r-1).n! combinations where n is the number of machines. A large number of calculations cannot provide complete safety against backtrack so it is not worth it.

A flow chart for the calculation of Δ is given on the following page. The number of searches for the first placing can be cutdown to only those operations which are in clash with an operation having the earliest finish time.

All operations in the following chart refer to the unassigned operations.


Choose a machine k
Calculate for all job i,
Ti = Sum of the durations of operations on job i that follow the operation, ik
Find the operation, iok having the earliest finish time
Is an operation ik in clash with the operation iok ?
No
Δ2 = ∞
Choose a remaining operation ik.
Yes
Consider the operation ik for the first placing
Out of the remaining operations on machine k, choose an operation, say jk, to be placed second. Then let, DT = completion time
DT = Completion time of the set of operations ik and jk.
Calculate
$\Delta^2_{ij} = \sum_{l \neq i,j} \{ DT + D_{lk} + T_l - SJ_l, 0.0 \}$
Are all operations on machine k accounted for the second place?
No
Yes
Are all operations accounted for first placing?
No
Yes
Δ^k_2 = minimum of all Δ^2_{ij}'s
exit

Then the Δ is chosen to be the maximum of Δ_2^k for all k machines. The bound will then be calculated as

$$Z = \sum_{i=1}^{m} CJ_i \cdot SJ_i + \sum_{k=1}^{m} CM_k \cdot SM_k + \Delta \qquad (16)$$

A job-shop problem using the above bounds has been solved in the article 5.of this chapter.

Before closing this chapter it should be observed that the branch and bound procedure is a powerful tool to solve the job-shop scheduling problem. More than that it gives considerable insight into such problems.

CHAPTER IV

A SUBOPTIMAL BRANCH AND BOUND PROCEDURE

In the previous chapter we have seen that to eliminate backtracking one has to manipulate a large amount of data. Even with the restricted bounds proposed, the method will require considerable computer time for large problems. In large problems the effect of scheduling an operation will give relatively small increase in in the total cost. The larger the difference between this cost and the cost of the optimal schedule, the larger the backtracking steps that would be expected. Unfortunately even ordinary industrial problems are too big for the optimal methods proposed in the last chapter. This is the reason industries have not shown a keen interest in the exact solution of the scheduling problem. Therefore, there is a great need of finding some quick suboptimal procedures. In the following discussion a suboptimal procedure is developed on the lines of the branch and bound procedure developed in the previous chapter.

4.1 SUBOPTIMAL METHOD

The procedure developed in the previous chapter constructs the schedule by proceeding from left to right resolving the clashes according to the lowest bound criteria. This is also the natural way any scheduler will adopt even though he may not be knowing anything

about the branch and bound method. In place of the selection according to the lowest bound, he often uses thumb rules or heuristic rule like 'first come first served' or 'shortest imminent operation (SIO)' rule. The heuristically generated schedules are generally one pass and no backtracking steps are undertaken. Now that we have bounding procedure for any linear cost function we can also go for one pass solutions. Therefore the method developed in the previous chapter, with all backtracking steps ignored, can easily replace the heuristic and thumb rules.

Note that in the bounding procedure we have considered the effect of next two placings of the unassigned operations on all machines. This is the 'look ahead' feature of this procedure. The question that we face is, whether this is a good guarantee for obtaining a near optimal solution. It does not give any such guarantee. It assures us only of a locally optimum solution. In any case its guarantee for a locally optimum solution is far better than by any existing heuristic or thumb rule because they do not have 'look ahead' property.

4.2 CRITICAL OPERATIONS

Giffler et.al.(1) have found that with the increase in the range of durations of various operations, the combinatorial aspects of the problem also increase. This is expected because such problems have higher possibility

While resolving clashes among two or more operations on a machine, it is quite possible that bounds for some nodes have same values. This tie can be resolved by selecting an operation which gives larger push to other operations in the schedule. It may also happen that for an operation, which gives larger rightward push to other operations, the bound is only slightly higher than the lowest bound. Since this operation is likely to ease more clashes among other operations it can be a critical operation. The selection according to numerically lowest bound will ignore it. A method to take care of such critical operations can be suggested as follows :

Choose those nodes which have bounds within the lowest bound plus the increment in the lowest bound over the lowest bound of the immediately preceding branch of the assignment tree. Calculate indices by dividing the increase of each of these bounds over the lowest bound of the previous branch by the displacement that its corresponding operation causes to other operations. Choose the operation corresponding to the lowest index.

Let us consider the job-shop problem as given in the Fig.1. Suppose the bounding procedure restricts itself to finding the Δ due to placing of only one of the 'next' operations on each machine, as against suggested two placings in article 6 of chapter III. The partial tree for the problem will be as follows :

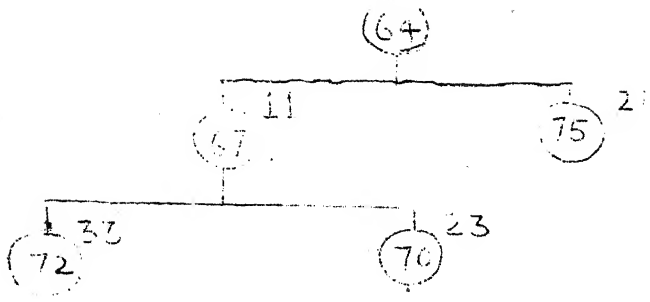


Fig 2

The displacement caused by placing operation 33 in the first place on the machine 3 shifts operation 13 by 1 and operation 23 by 5. The total displacement caused is 6. The total displaceement caused by operation 23 is 1. Then the indices are,

$$\text{index for operation 33} : \frac{72 - 67}{6} = \frac{5}{6}$$

$$\text{index for operation 23} : \frac{70 - 67}{6} = 3$$

The index is lower for the operation 33 eventhough its bound value is numerically higher than that of operation 23. Note theat a displacement of 6 by operation 33 does not give relatively as higher a increase in the bound as the operation 23 given with the displacement of only 1 unit. Therefore we decide on 33 as the critical operation and schedule it. The resulting schedule after this choice gives a total cost of 17 which is optimal, while that corresponding to the choice of operation 23 gives a total cost of 20. The Gnatt charts are as follows.

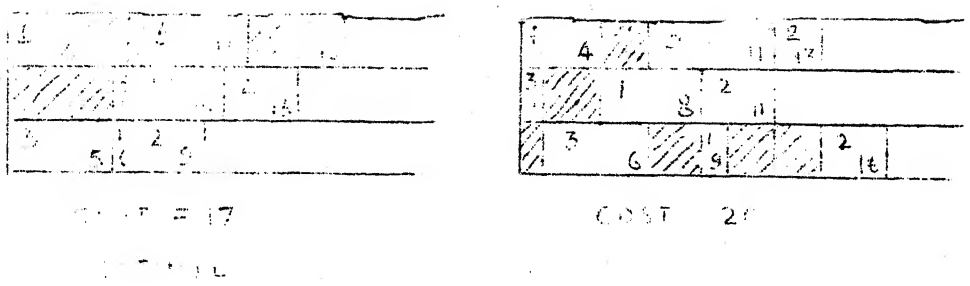


Fig.3

4.3 COMPARISON

A comparison between the suboptimal procedures with and without the facility of recognizing the critical operation given in Table.4.1. The earlier procedure is referred to as procedure 2 and the later as procedure 1. The bounding procedure is the same as given in article 6 of the Chapter III.

The problems, selected for comparison are such that the differences in the durations are large, which increases the combinatorial aspects of the problems. There are two types of problems for each size; one of which has identical sequences of processing on all jobs and the other has non-identical sequences. The later further increases the combinatorial aspects of the problem. The costs of unit job late time and machine idle-time are alike and unity. The problems are given in Appendix 1.

The results listed in the table are the costs of the overall schedules. We can see that the results obtained by procedure 1 are in general better than those obtained by procedure 2. This shows that the scheme used to detect the

critical operations is not successful. Some more investigation is required in this direction.

The time taken by IBM 7044 to solve all the problems in Table 4.1 is nearly 280 seconds for both procedures 1 and 2. Out of this time nearly 100 seconds are used in compiling the program.

Table 4.1 also compares the results of procedures 1 and 2 with the one produced by using heuristic rule SIO (Shortest Imminent Operation Rule). According to this rule all those operations in the mix are scheduled which use different machines and are completed in earliest time. This rule has been found to be very efficient for minimizing the total time of completion of all jobs(1). Here we have calculated the costs of both joblate-times and machine idle times for the schedule generated by using SIO.

We observe that all the results obtained by both suboptimal procedures 1 and 2 are better than that obtained by using SIO. In case the costs of job latetimes and machine idle-times are nonunity and random, the suboptimal procedures will be undoubtedly better than the SIO rule. This strengthens our belief in replacing the heuristic rules by the suboptimal procedure 1.

Problem size	Suboptimal procedure 1	Suboptimal procedure 2	SIO
1. 4x3 identical orders	53	56	56
2. 5x5 identical orders	223	208	342
3. 5x5 nonidentical orders	151	145	199
4. 6x6 identical orders	192	204	206
5. 6x6 nonidentical orders	215	220	247
6. 7x7 identical orders	699	765	799
7. 7x7 nonidentical orders	392	418	467
8. 8x8 identical orders	1019	1007	1347
9. 10x10 nonidentical orders	6932	7621	7662

The cost of schedule

Table 4.1

4.3 COMPUTER PROGRAM

A computer program has been written in FORTRAN IV for the suboptimal procedure 1 which has no facility of recognising the critical operations. The program consists of one main program and three subroutines. The subroutine SUB_2 constructs the Gnattchart for any node. The SUB_3 calculates the bounds in the Gnatt chart produced in SUB_1 according to the procedure given in article 6 of chapter III. The SUB_1 finds out an unassigned operation on a machine which has least time of completion with Gnatt chart. This is used to find the clashing operaions on a machine at any stage of scheduling process.

In the main program the mix is formed and updated as the scheduling proceeds. If an operation in the mix is not in clash it is directly scheduled without having to call the subroutines. The main program decides upon a proper machine in the mix and calls the subroutines to construct the Gnatt charts and calculate the bounds for for the clashing operations on the selected machine. A decision is taken according to the lowest bound and again the procedure is repeated.

The format statements for the input and output of data must be provided by the user. The data arrangement is as follows :

(1) Number of jobs and number of machines in I4 format on a single card.

(2) Input format specification for the order matrix. If the matrix is 10x6 where 10 is the number of jobs, then a format must be specified as (6I4) starting from the first column of an IBM card.

(3) Format specification for the durations matrix. If the matrix is 10 x 6, then a format like (6F6.1) must be specified on a single card starting from the first column. The same format is also used for the output of the scheduled durations matrix.

(4) Format specification for supplying the information about the arrival time of jobs and the cost per unit joblatetimes. The format should be specified as

(F6.1, F 0.4) on a single card. The number of columns punched should not exceed 24. The first format refers to the arrival time of jobs.

(5) Input data for order matrix.

(6) Input data for the durations matrix.

(7) Input data for the arrival time of jobs and the costs of joblatetimes. The order of punching these data on a single card must be the same in which they are mentioned here. Data for each job is punched on separate card.

(8) Input data for the available times on machines and the costs of machine idletimes must be punched in the same manner as mentioned in (7).

The output of the scheduled matrix is arranged in the form of a matrix, the rows of which are the jobs and the columns indicate the machines. The (i,j)th entry in the matrix is the scheduled start time of the operation of ith job on jth machine.

4.4 ESTIMATION OF THE COMPUTER TIME

In combinatorial problem like the job-shop scheduling it is not possible to give a closed form relationship between the size of the problem and the time required to find the suboptimal solution. It all depends on how the various operations on different jobs

fit with each other. The uncertainty increases with the increase in range of differences in the processing times. We can estimate the time taken for the worst possible case in which we have to resolve clashes among all the jobs on all the machines. If there are m jobs and n machines, then we have to calculate at worst $(m-1)!-1).n$ nodes.

The calculation of nodes in the beginning of the schedule requires maximum manipulation of a large data while those at the end requires the minimum. The average calculations involved in a node which is in the middle can be estimated as follows :

Calculation of SM_k $(k= 1,\ldots n/2)$: $\frac{n}{2}$ DO loops of length $\frac{m}{2}$. So $\frac{nm}{4}$ additions.

Calculation of $_k$ $(k = 1,\ldots n/2)$: $\frac{m}{2}(\frac{m}{2}-1)(\frac{m}{2}-2).\frac{n}{2}$ additions.

Construction of Gnatt chart : $(\frac{m}{2}-1)$ shifts or additions.

The various tests may be expected to take as much time as all the above additions will take. Thus a very very crude estimation of the equivalent number of additions will be,

$$2(\frac{mn}{4}+\frac{nm^3}{8}-\frac{3m^2n}{4}+nm+\frac{m}{2}-1)$$

Generally m is very large compared to n, then this number can be approximated as

$$n\,m^3/4$$

Therefore for all the nodes approximately $\frac{n^2m^2}{4}(m!)$ number of equivalent additions are required to be done.

This number is very strongly dependent upon the number of jobs. This number is very crude because it is very unlikely that all the unassigned operations simultaneously clash over all machines. However, we can expect that this maximum estimate should have proportionate relationship with the actual time of computation.

The estimation of minimum time of computation is very difficult for general problem in which there is nonidentical orders of processing various jobs on different machines. For the identical-order case the minimum number will be (m-1)! nodes, The time required for calculation of average node will be nearly same as in the worst case. But this number is not so informative as the maximum number calculated for the worst case. When jobs increase from m to m 1, the increase in the cost of computation can be found as follows

$$\triangle \text{Cost} \propto C.\ \left\{(m+1)!\ \frac{n^2(m+1)^2}{4} - (m)!\ \frac{n^2m^2}{4}\right\}$$

where C is a constant.

Further simplification gives

$$\triangle \text{Cost} \propto C.\ \frac{n^2m^2}{4}\, m!\ \left\{\frac{(m+1)^3}{m^2} - 1\right\}$$

$$\approx \frac{n^2m^2}{4}\, m!\ .\ m$$

that is, Δcost $\propto$ cost of previous schedule multiplied by m.

This gives us the cost of computation as an exponential function of the square of the number of jobs when the number of machines in the shop is constant.

CHAPTER V

INCLUSION OF SOME PRACTICAL SITUATIONS

The optimal branch and bound method and the sub-optimal method, both have linear objective functions. In general the objective function is nonlinear, for example the due date problem is very common. Likewise there are many other constraints like assembly and disassembly, which occur almost without fail in all industrial activities. Besides, very often new job orders arrive and what should be the managements reaction twoards it? All these are very practical problems. A few suggestions to take account of such situations are given in the following discussion .

5.1 ARRIVAL OF A NEW JOB ORDER

A new job may arrive while a batch of jobs is still under processing. The management is faced with the problem whether the present batch should be disturbed or not. If the new job is a rush order then the present batch have to be disturbed. In this case the old schedule has to be scrapped completely and a new branch and bound solution must be generated from the arrival time of the new job. In case there is planning ahead then one can think of disturbing the schedule even prior to the arrival time of the new job. It all depends on the expected profit from the new job. From the previous chapter we know that the cost of computation is an exponential function of the

number of jobs. The estimated increase in the cost of computation involving a new job will be,

Δ Cost computation = Cost of computation of original schedule multiplied by the number of jobs in the previous schedule.

The profit of scheduling a new job can be estimated as follows :

Fit the new job to the existant schedule without disturbing it and calculate the cost of the new schedule. Let it be C_1.

Suspend scheduling all the operations from a point from which the new job disturbs the machine idle-times of the previous schedule. Form the mix of next immediate operations at that instant and calculate a node corresponding to giving priority to the new job. Let the resulting node value be C_2.

If the cost of the previous schedule be C, then $C_2 - C$ is the minimum increment in the cost. The difference between $C_2 - C$ and $C_1 - C$ is an estimate of the profit from from the new job. If this compares favourably with the increment in the cost of computation, then we should re-schedule otherwise not.

5.2 DUE DATES

In these problems the objective function is piecewise linear. If the constraints of the LPP are linear then the region of the solution will still be a convex polyhedron or r dimensions where r is the number of variables. The intersection of the objective function with this polyhedron will be a polyhedron of size r or greater dimensions as against a polyhedron of r-1 size in case of linear objective function. Integerization of any variable will be equivalent to parallel shifting of the intersection polyhedron. If the integer takes values of 0 or 1 only, then the objective function corresponding to zero or one will constitute the lower bound for the objective function. Hence the branch and bound procedure can still be applied to this case. This technique is a very general search technique.

The bounding procedure given in the third chapter can be used for the above problem. The difference will be only in selecting the new machine to be considered for resolving clashes. A good selection criteria can be the one which has the highest derivative of the cost function.

5.3 ASSEMBLY AND DISASSEMBLY OF JOBS

The assembly of two or more jobs can be considered as yet another job which has its earliest starting time dependent on the time of finishing of all the assembled jobs. If job i and j are assembled, then for the assembled

job l we can add the following constraint in the set of equations

$$S_{lk} \geq S_{ik_1} + D_{ik_2} \qquad (1)$$

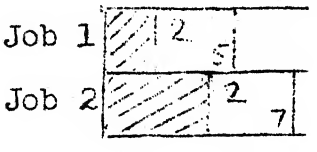
$$S_{lk} \geq S_{jk_2} + D_{jk_2} \qquad (2)$$

The constraints will be taken care of while constructing the Gnatt chart.

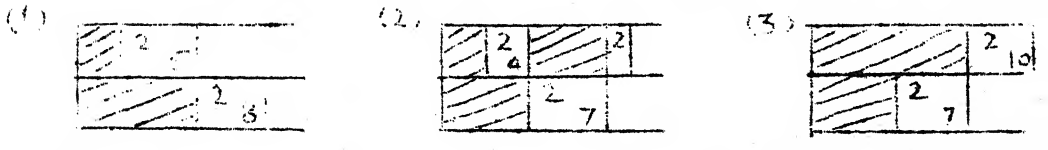
The cost of joblateness is assigned to the assembled job and not the constituent parts.

5.4 INTERRUPTION OF OPERATIONS

If we allow interruption then the number of nodes which must be calculated increases. Consider the following two jobs in clash over the kth machine.



The resolution of clash can be done in the following ways



The second possibility is a creation of the interrupting facility. The bounds are calculated for all these possibilities and the one with the lowest value of the bound is chosen. If the number of jobs in simultaneous clash is more than two the possibilities will increase, so also the time of computation. The bounds must also take into account the interruptibility of operations while calculating SM_k's and the incremental costs.

5.5 MORE THAN ONE MACHINE OF A KIND

This is a very practical situation to occur in any type of industry. There may be more than one lathe of the same kind in a workshop. Just like the case of interrupted operations it also increases the number of nodes that must be calculated in order to find the lowest bound.

The node calculations will be slightly different than that given in Chapter III. The SM_k's will now be calculated as the minimum time of finishing the last operation on the group of machines of kind k, taking into account all the possibilities of simultaneous operation of various jobs on these machines. Since this will require consider search time it will be advisable to consider the time of finishing the last operation on the kth machine in the present partial schedule.

The calculation of incremental costs due to placing of next two unassigned jobs on the kth machine will also become cumbersome. The incremental costs thus obtained will also be of verylittle help because a lot of compactness in the schedule is to be expected. Therefore, the calculation of bounds becomes very simple.

5.6 ALTERNATE ROUTES

Some jobs may have alternate routes of processing on various machines. Note that in the fixed-route case the SJ_i's, the time of finishing the last operation on the

ith job, is fixed. In this case the SJ_i's will be calculated to be the minimum time of finishing among the alternate routings of the unassigned operations on the ith job. The various combinations of routes will complicate the calculation of SM_k and also the incremental costs due to inter-job interactions. Therefore, as suggested in the article 5.5, the calculation of incremental costs is suspended altogether and the SM_k are chosen to be the time of finishing the last scheduled operation on the kth machine plus the sum of the processing times of the remaining operations on this machine.

5.7 CONCLUSION

The search procedure given in the Chapters III and IV can incorporate many practical situations with very little change.

APPENDIX I

SAMPLE PROBLEMS

1. 1(3) 2(10) 3(4)
   1(4) 2(1) 3(5)
   1(4) 2(8) 3(6)
   1(6) 2(4) 3(1)

2. 1(3) 2(6) 3(9) 4(1) 5(8)
   1(5) 2(16) 3(6) 4(8) 5(10)
   1(9) 2(2) 3(7) 4(10) 5(15)
   1(3) 2(9) 3(14) 4(2) 5(15)
   1(1) 2(8) 3(25) 4(22) 5(3)

3. 3(3) 2(6) 5(9) 1(1) 4(8)
   4(5) 1(16) 2(6) 3(8) 5(10)
   5(9) 2(2) 4(7) 1(10) 3(15)
   4(3) 1(9) 2(14) 3(2) 5(15)
   2(1) 3(8) 4(25) 5(22) 1(3)

4. 1(1) 2(3) 3(6) 4(7) 5(3) 6(6)
   1(8) 2(5) 3(10) 4(10) 5(10) 6(4)
   1(5) 2(4) 3(8) 4(9) 5(1) 6(7)
   1(5) 2(5) 3(5) 4(3) 5(8) 6(9)
   1(9) 2(3) 3(5) 4(4) 5(3) 6(1)
   1(3) 2(3) 3(9) 4(10) 5(4) 6(1)

5. 3(1) 1(3) 2(6) 4(7) 6(3) 5(6)
2(8) 3(5) 5(10) 6(10) 1(10) 4(4)
3(5) 4(4) 6(8) 1(9) 2(1) 5(7)
2(5) 1(5) 3(5) 4(3) 5(8) 6(9)
3(9) 2(3) 5(5) 6(4) 1(3) 4(1)
2(3) 4(3) 6(9) 1(10) 5(4) 3(1)

6. 1(12) 2(3) 3(15) 4(9) 5(5) 6(6) 7(2)
1(3) 2(6) 3(21) 4(8) 5(4) 6(12) 7(10)
1(8) 2(28) 3(6) 4(40) 5(2) 6(8) 7(9)
1(7) 2(25) 3(14) 4(6) 5(9) 6(1) 7(13)
1(26) 2(9) 3(3) 4(12) 5(5) 6(24) 7(7)
1(5) 2(12) 3(6) 4(16) 5(9) 6(34) 7(7)
1(12) 2(16) 3(8) 4(7) 5(19) 6(6) 7(3)

7. 1(12) 2(3) 3(15) 4(9) 5(5) 6(6) 7(2)
3(3) 2(6) 6(21) 5(8) 1(4) 4(12) 7(10)
4(8) 1(28) 7(6) 6(40) 2(2) 3(8) 5(9)
5(7) 2(25) 3(14) 4(6) 6(9) 7(1) 1(13)
4(26) 1(9) 6(3) 7(12) 3(5) 2(24) 5(7)
2(5) 7(12) 4(6) 5(16) 1(9) 3(34) 6(7)
7(12) 6(16) 5(8) 4(7) 2(19) 3(6) 1(3)

8. 1(2) 2(13) 3(24) 4(10) 5(5) 6(6) 7(9) 8(10)
   1(36) 2(6) 3(15) 4(5) 5(9) 6(3) 7(8) 8(24)
   1(5) 2(25) 3(23) 4(6) 5(38) 6(9) 7(3) 8(2)
   1(9) 2(12) 3(13) 4(35) 5(6) 6(9) 7(1) 8(3)
   1(5) 2(3) 3(19) 4(1) 5(5) 6(12) 7(15) 8(3)
   1(9) 2(13) 3(5) 4(18) 5(9) 6(17) 7(20) 8(30)
   1(5) 2(3) 3(2) 4(25) 5(36) 6(9) 7(1) 8(7)
   1(6) 2(9) 3(19) 4(4) 5(22) 6(6) 7(7) 8(28)

9. 1(29) 2(78) 3(9) 4(36) 5(49) 6(11) 7(62) 8(56) 9(44) 10(21)
   1(43) 3(90) 5(75) 10(11) 4(69) 2(28) 7(46) 6(46) 8(72) 9(30)
   2(91) 1(85) 4(39) 3(74) 9(90) 6(10) 8(12) 7(89) 10(45) 5(33)
   2(81) 3(95) 1(71) 5(99) 7(9) 9(52) 8(85) 4(98) 10(22) 6(43)
   3(14) 1(6) 2(22) 6(61) 4(26) 5(69) 9(21) 8(49) 10(72) 7(53)
   3(84) 1(2) 6(52) 4(95) 9(48) 10(72) 1(47) 7(65) 5(6) 8(25)
   2(46) 1(37) 4(61) 3(13) 7(32) 6(21) 10(32) 9(99) 8(30) 5(55)
   3(31) 1(86) 2(40) 6(74) 5(32) 7(88) 9(19) 10(48) 8(36) 4(79)
   1(76) 2(69) 4(76) 6(51) 3(65) 10(11) 7(40) 8(89) 5(26) 9(74)
   2(85) 1(13) 3(61) 7(7) 9(64) 10(76) 6(47) 4(52) 5(90) 8(45)

COMPUTER PROGRAM

```
JOB [illegible] 130,TI[illegible],[illegible]ES[illegible],NAME  JOB SHOP SCHEDULING
INJ[illegible]
INFTC
      DI[illegible]SI[illegible] [illegible](10,10),TIME(10,10),COSTJ(10),COSTM(10),OPT(10,10),
     1SEDUL(1[illegible]10), [illegible]Y(10,10),DM(10),DJ(10),ATIME(10),DMD(10),DJD(10)
     1,NFLAG(1 ,10) ,D[illegible]G(10),DJG(10),DUM(10,10)      ,MJ(10,10)
     1 [illegible](10),LCNT(10),MIX(10),DSM(10),ARRVT(10),FM3(4),AVLTJ(10)
     1 ,DS[illegible](10),D[illegible]P(10),K[illegible]P(10),FM1(2),FM2(2) ,NUM(10)
      CO[illegible] /A/ K1,K2,K3,TIME,JORDR,SEDUL,ATIME,DJ,DM,M,N ,NEXT,NFLAG
     1,OPT,COST,COSTM,COSTJ,MJ,NUM
      MG,JK,J[illegible] ETC ARE THE  FLAGS
28     FORMAT(* TOTAL COST =  *,F10.4)
720   FORMAT(* [illegible] OF NODES CALCULATED = *,I4)
2[illegible]     FORMAT(* A[illegible]-1.0[illegible] IN (I,J)TH POSITION INDICATES THAT NO SUCH OP
     1WAS SPECIFIED BY THE USER*)
94[illegible]   FORMAT(    *MACHINES IN COLUMNS AND JOBS IN ROWS*)
721   FORMAT(1X,*SCHEDULE*)
[illegible]6     FORMAT(1X,*AVLTM COSTM*)
65    FORMAT(1X,*ARRVT COSTJ*)
64     FORMAT(1X,*DURATION MATRIX*)
[illegible]3     FORMAT(1X,*ORDER MATRIX*)
32     FORMAT(1X,*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX*)
2     FORMAT(4A6)
      K1=0
      K2=0
      K3=0
      N-THE NUMBER OF JOBS
      M-THE NUMBER OF MACHINES IN THE SHOP
      READ1,M,N
      READ2,(FM1(I),I=1,2)
      READ2,(FM2(I),I=1,2)
       READ2,(FM3(I),I=1,4)
      NODES=0
       COST-ESTIMATED COST OF THE EXISTANT SCHEDULE
      JORDR-SEQUENCE MATRIX AS GIVEN  BY USER
      TIME-DURATIONS
      COSTJ-COST OF JOB LATENESS  PER UNIT TIME
      COSTM-COST OF MACHINE IDLENESS PER UNIT TIME
      ATIME-AVAILABLE TIME ON  MACHINES
      ARRVT-ARRIVAL TIMES OF JOBS
      AVLTJ- EARLIEST TIME WHEN A JOB IS FREE
```

```
      COST=0.
       READ(5,FM1)((JORDR(I,J),J=1,N),I=1,M)
      PRINT82
      PRINT83
      WRITE(6,FM1)((JORDR(I,J),J=1,N),I=1,M)
       READ(5,FM2)((TIME(I,J),J=1,N),I=1,M)
      PRINT84
      WRITE(6,FM2)((TIME(I,J),J=1,N),I=1,M)
       PRINT85
       READ(5,FM3)(ARRVT(I),COSTJ(I),I=1,M)
       WRITE(6,FM3)(ARRVT(I),COSTJ(I),I=1,M)
       READ(5,FM3)(ATIME(J),COSTM(J),J=1,N)
      PRINT86
      WRITE(6,FM3)(ATIME(J),COSTM(J),J=1,N)
771            DO 400 J=1,N
      DM(J)=0.0
      MK(J)=0
       DO 400 I=1,M
400   MJ(I,J)=0
      FORMATION OF EARLIER MIX    --------------- - +
      NUM- NUMBER OF OPERATIONS ON A JOB
      OPT-DURATIONS OF OPS ARRANGED MACHINEWISE
      SEDUL-SCHEDULED START -TIMES
      DM -ESTIMATED TIME OF FINISHING THE LAST OPERATION ON MACHINES
      DJ -ESTIMATED TIME OF FINISHING THE LAST OPERATION ON JOBS
  NFLAG-A ZERO IN (I,J)TH POSITION SHOWS THAT THE OPERATION IS NOT SCHE
    AND A ONE  SHOWS THAT IT IS SCHEDULED. A NEGATIVE NUMBER SHOWS NO O
      MJ-(I,J)TH POSITION INDICATES THE ORDER OF  THE IJ,TH OP ONTHE ITH
       MK-MK(J) INDICATES THE NUMBER OF OPS ON MACHINE J IN THE MIX
      MIX-THE IMMEDIATELY NEXT UNASSIGNED OPS ON ALL JOBS
      A ZERO IN IT SHOWS THAT ALL OPS ON THE JOB ARE OVER
      DO 301 I=1,M
      NUM(I)=N
      MIX(I)=I*100+1
      J1=JORDR(I,1)
      MK(J1)= MK(J1)+1
      DO 4 J=1,N
      NFLAG(I,J)=0
      J1=JORDR(I,J)
      IF(J1.EQ.0) GO TO 961
      OPT(I,J1)=TIME(I,J)
      MJ(I,J1)=J
      IF(J-1)5,5,6
6     SEDUL(I,J1)=SEDUL(I,J0)+TIME(I,J-1)
      GO TO 4
961     NUM(I)=J-1
      GO TO 4
5     SEDUL(I,J1)=AMAX1(ARRVT(I),ATIME(J1))
4     J0=J1
      DO 7 JV=1,N
```

```
      IF(JO(I,JV).NE.0) GO TO 7
      SEDUL(I,JV)=-1.0
      NFLAG(I,JV)=-1
 7     CONTINUE
 301  CONTINUE
C     ---------------------------------------------    -
C     CALCULATION OF THE [illegible] COST   [illegible]
        CALL [illegible](COST,[illegible],SEDUL)
      NODES=NODES+1
C     SEARCH OF AN OP IN THE MIX WHICH IS NOT IN CLASH ----
 773   JM=1
      DO 772  I=1,M
      IF(MIX(I).EQ.0) GO TO 772
      JM=2
      K10=MIX(I)
      KK1=K10/100
      KK2= K10-KK1*100
      KK3=JORDR(KK1,KK2)
      DO 43 L=1,M
      IF(L.EQ.KK1) GO TO 43
      IF(NFLAG(L,KK3).NE.0) GO TO 43
      IF((SEDUL(KK1,KK3)+OPT(KK1,KK3)).GT.SEDUL(L,KK3)) GO TO 772
 43    CONTINUE
      KMI =K10
        GO TO 45
 772  CONTINUE
      IF(JM.EQ.1) GO TO 26
C     ---------------------------------------------  -
C     SELECTION OF THE PROPER MACHINE IN THE MIX  ---------
C     DSM- MINIMUM DISPLACEMENT CAUSED ON A MACHINE BY AN OP IN THE MI
       DO 912 J=1,N
 912  DSM(J)=0.0
      DO 910 IZ=1,M
      IF(MIX(IZ).EQ.0) GO TO 910
      LC1=MIX(IZ)/100
      LC2=MIX(IZ)-LC1*100
      LC3=JORDR(LC1,LC2)
      CM=0.0
      DO 911 IY=1,M
      IF(IY.EQ.IZ) GO TO 911
      IF(NFLAG(IY,LC3).NE.0) GO TO 911
      IF(SEDUL(IZ,LC3).GE.SEDUL(IY,LC3)+OPT(IY,LC3)) GO TO 910
 951  CM=CM+AMAX1( SEDUL(IZ,LC3)+OPT(IZ,LC3)-SEDUL(IY,LC3),0.0)
 911  CONTINUE
      IF(DSM(LC3).EQ.0.0) DSM(LC3)=CM
      DSM(LC3)=AMIN1(DSM(LC3),CM)
 910  CONTINUE
C     LCNT  -DUMMY OF MK
      DO 760 J=1,N
 760  LCNT(J)=MK(J)
```

```
 770  COTM=.0
      JC=0
      DO 763  J=1,M
      IF(LCNT(J).  ) GO TO 763
      IF(C    .GE. DM(J)) GO TO 763
      C TM=DS (J)
      JC=J
 763  CONTINUE
      IF(JC.EQ.0) GO TO 26
C     ------------------ ---------------------  -
C        FINDING THE LOWER BOUND  --------------+-
 774  KNT=1
      CALL DALLOW(SEDUL,DALOW,JC)
      K8=0
      DO 765  I=1,M
      IF(MFLAG(I,JC).NE.0) GO TO 765
      IF(SEDUL(I,JC).GT.DALOW) GO TO 765
     K10=I*10+MJ(I,JC)
      K8=K8+1
C      DUMMY-DUMMY OF SEDUL
C      DMD AND DJD ARE DUMMYS OF DM AND DJ RESPECTIVELY
C      DELS- DIFFERENCE BETWEEN A BOUND AND THE LOWEST BOUND OF THE
C     BRANCH IN THE ASSIGNMENT TREE
C      DSMK-        DISPLACEMENT IN OTHER OPS BY THE OP CORRESPONDING
 7     CALL GNATT(DELS,K10,DUMMY,DMD,DJD,DSMK(K8))
C      DMP-BOUNDS FOR VARIOUS NODES
C      KMP-LABELS OF NODES
      KMP(K8)=K10
      DMP(K8)=DELS
      NODES=NODES+1
      IF(KNT.GT.1) GO TO 410
      KNT=2
C      STORING GNATT-CHART CORRESPONDING TO LOWER BOUND
 413  KMIN=K10
      K17=K8
      IC=I
      SMIN=DELS
C      SMIN- MINIMUM DELS
C      DUM-DUMMY OF SEDUL
C      DJG AND DMG ARE THE DUMMYS OF DJ AND DM
       DO 635 J=1,N
       DMG(J)=DMD(J)
      DO 635 II=1,M
 635  DUM(II,J)=DUMMY(II,J)
      DO 637  II=1,M
 637  DJG(II)=DJD(II)
       IF(SMIN.GT.0.0) GO TO 765
      IF(   MIX(IC)-K10) 415,15,415
 410  IF(DELS)766,766,801
 766  IF(MIX(K1)-K10) 415,413,415
```

```
[illegible] IF(S[illegible]T-[illegible]LC)7[illegible], [illegible]2,413
415   LC[illegible]T(J[illegible])=[illegible]
      GO TO 77[illegible]
[illegible]2    IF([illegible]IX([illegible]3).[illegible]UI[illegible]) GO TO 765
      IF([illegible]IX(IC).N[illegible] [illegible]IN) GO TO 41[illegible]
      IF(DSM[illegible](K17)-DS[illegible](K8)) 413,76[illegible],765
7[illegible]5    CONTINUE
      IF(S[illegible]T[illegible]0.1) GO TO 26
C     ----------------------------------------------------------------
C     SCHEDULING OF THE SELECTED OPERATION   ----------
15    IF(MIX(IC).[illegible]KMIN) GO TO 415
846   DO 636 J=1,N
      D[illegible](J)=DM[illegible](J)
      DO 636 I=1,M
636   SEDUL(I,J)=DU[illegible](I,J)
      DO 638 I=1,M
638   DJ(I)=DJ[illegible](I)
921   DO  920 I=1,K8
920   DMP(I)=DMP(I)+COST
      COST=COST+SMIN
45     L1=KMIN/100
      L2=KMIN-L1*100
      L3=JORDR(L1,L2)
      MFLAG(L1,L3)=1
      MK(L3)=MK(L3)-1
      IF((L2+1).GT.N)            GO TO 775
      KP=KMIN+1
      KP1=KP/100
      KP2=KP-KP1*100
      KP3=JORDR(KP1,KP2)
      IF(KP3.EQ.0) GO TO 775
      MIX(KP1)=KP
      MK(KP3)=MK(KP3)+1
      GO TO 773
775   MIX(L1)=0
      GO TO 773
C     ----------------------------------------------------------------- -
26     PRINT721
      PRINT940
      PRINT 29
      WRITE(6,FM2)((SEDUL(I,J),J=1,N),I=1,M)
640   SUMD=0.0
      DO 490 I=1,M
      DO 490 J=1,N
490   SUMD=SUMD+OPT(I,J)*(COSTJ(I)+COSTM(J))
      COST=COST-SUMD
      PRINT28,COST
      PRINT720,NODES
      STOP
      END
```

```
$IBFTC SUB1
      SUBROUTINE GRATT(DELS,K10,DUMMY,DMD,DJD,DCP)
      DIMENSION JORDR(10,10),TIME(10,10),COSTJ(10),COSTM(10),OPT(10
     1),SEDUL(10,10), DUMMY(10,10),DM(10),DJ(10),ATIME(10),DMD(10),DJ
     1,NFLAG(10,10),LCNT(10),JORDR(10,10),MJ(10,10),DSM(10),DUM(10,
     1 ,DMP(10),TD(10),LCNTR(10),COSM(10),NUM(10)
      COMMON /A/ K1,K2,K3,TIME,JORDR,SEDUL,ATIME,DJ,DM,M,N,NEXT,NF
     1,OPT,COST,COSTM,COSTJ,MJ,NUM
      ALOW=0.0
       I=0
      DO 815 I=1,M
      IF(NFLAG(I,JC),NE,0) GO TO 815
      X=DUMMY(I,JC)+OPT(I,JC)
      IF(X),GT,0) GO TO 816
      K1=1
 817  XMIN=X
      IDEX=I
      GO TO 815
 816  IF(XMIN-X) 815,815,817
 815  CONTINUE
      IF( K1,NE,0) ALOW = DUMMY(IDEX,JC)+OPT(IDEX,JC)
      RETURN
      END


$IBFTC SUB2
      SUBROUTINE GRATT(DELS,K10,DUMMY,DMD,DJD,DCP)
      DIMENSION JORDR(10,10),TIME(10,10),COSTJ(10),COSTM(10),OPT(10
     1SEDUL(10,10), DUMMY(10,10),DM(10),DJ(10),ATIME(10),DMD(10),DJ
     1,NFLAG(10,10),LCNT(10),JORDR(10,10),MJ(10,10),DSM(10),DUM(10,
     1 ,DMP(10),TD(10),LCNTR(10),COSM(10),NUM(10)
      COMMON /A/ K1,K2,K3,TIME,JORDR,SEDUL,ATIME,DJ,DM,M,N ,NEXT,NF
     1,OPT,COST,COSTM,COSTJ,MJ,NUM
      DCP=0.0
      DO 302 I=1,M
      DO 302 J=1,N
 302   DUMMY(I,J)=SEDUL(I,J)
      DELS= 0.0
      K1=K10/100
      K2=K10-K1*100
      K3=JORDR(K1,K2)
      FTDUM=DUMMY(K1,K3)+TIME(K1,K2)
      NFLAG(K1,K3)=1
      DO 401 I=1,M
      IF((NFLAG(I,K3),NE,0),OR,(SEDUL(I,K3),GE,FTDUM)) GO TO 401
      DCP=DCP+FTDUM-DUMMY(I,K3)
       DUMMY(I,K3) =FTDUM
      J1=MJ(I,K3)+1
      IF(JORDR(I,J1),EQ,0) GO TO 401
      IF(J1,GT,N) GO TO 401
```

```
      K4=K3
      DO 401 J=J1,M
      J2=JORDR(I,J)
      IF(NFLAG(I,J2).LT.0) GO TO 401
      K=JORDR(I,J)
      DUMMY(I,J)=AMAX1(SEDUL(I,K),DUMMY(I,K4)+TIME(I,J-1))
      K1=K
401   CONTINUE
      CALL BOUND(DELS,DMD,DJD,DUMMY)
      DELS=DELS-COST
      NFLAG(K1,K3)=0
      RETURN
      END


$IBFTC SUB3
      SUBROUTINE BOUND(DELS,DMD,DJD,DUMMY)
      DIMENSION JORDR(10,10),TIME(10,10),COSTJ(10),COSTM(10),OPT(10,1
     1SEDUL(10,10), DUMMY(10,10),DM(10),DJ(10),ATIME(10),DMD(10),DJD
     1,NFLAC(10,10),LCNT(10),NORDR(10,10),MJ(10,10),DSM(10),DUM(10,1
     1 ,DMD(10),TD(10),LCNTR(10),COSM(10),NUM(10)
      COMMON /A/ K1,K2,K3,TIME,JORDR,SEDUL,ATIME,DJ,DM,M,N ,NEXT,NFL
     1,OPT,COST,COSTM,COSTJ,MJ,NUM
C       COSMX-MAXIMUM COSM(K)
C      TD- SUM OF THE DURATIONS OF THE REMAINING OPS ON AJOB
C      LCNTR- DUMMY OF NFLAG
C      COSM-DELTA INCREASE IN THE BOUND VALUE
       COSMX=0.0
       DELS= 0.0
       DO 600 I=1,M
       M1=NUM(I)
       JT=JORDR(I,M1)
       DJD(I)=DUMMY(I,JT)+OPT(I,JT)
       DELS=DELS+COSTJ(I)*DJD(I)
  600  CONTINUE
C       CALCULATION OF SMK,THETIME OF FINISHING LAST OP ON A MACHINE
       DO 470 K=1,N
         DMD(K)=DM(K)
       DO 523 I=1,M
  523  LCNTR(I)=NFLAC(I,K)
  685  DO 525 I=1,M
  525  IF(LCNTR(I).EQ.0) GO TO 527
       DELS=DELS+COSTM(K)*DMD(K)
       GO TO 470
  527  DURAT=0.0
  526  JD=1
       DO 460 I=1,M
       IF(LCNTR(I).NE.0) GO TO 460
  686  IF(JD.NE.1) GO TO 462
       IC=I
```

```
      JD=2
      DMIN=DUMMY(I,K)
      GO TO 460
  462 IF(DUMMY(I,K).GE.DMIN) GO TO 460
      IC=I
      DMIN=DUMMY(I,K)
  460 CONTINUE
  737 IF(JD.EQ.1) GO TO 682
      LCNTR(IC)=1
      DURAT=AMAX1(DURAT,DUMMY(IC,K))+OPT(IC,K)
      DO 511 I=1,M
      IF(LCNTR(I).NE.0) GO TO 511
      IF(DUMMY(I,K)-DURAT) 735,735,511
  735 LCNTR(I)=1
  324 DURAT=DURAT+OPT(I,K)
  511 CONTINUE
      DMD(K)=AMAX1(DMD(K),DURAT)
      GO TO 526
C     ------------------------------------------
  682 JK=1
      DELS=DELS+COSTM(K)*DMD(K)
C     CALCULATION OF DELTA INCREMENT IN BOUND VALUE ----------
C     CALCULATION OF THE SUM OFDURATIONS OF THE REMAINING OPS ONTHE JOBS
      DO 705  I=1,M
      TD(I)=0.0
      IF(((I.EQ.K1).AND.(K.EQ.K3)).OR.(NFLAG(I,K).NE.0)) GO TO 705
       J2=MJ(I,K)+1
      IF(J2.GT.N) GO TO 705
      DO 481 LJ=J2,N
  481 TD(I)=TD(I)+TIME(I,LJ)
  705 CONTINUE
       COSM(K)=0.0
      CALL DALLOW(DUMMY,DALOW,K)
      DO 480 LI=1,M
      IF(NFLAG(LI,K).NE.0) GO TO 480
       IF(DUMMY(LI,K).GE.DALOW) GO TO 480
  690 DT=DUMMY(LI,K)+OPT(LI,K)
      NFLAG(LI,K)=1
      DO 900 LL=1,M
       IF(NFLAG(LL,K).NE.0) GO TO 900
      DT1=AMAX1(DT,DUMMY(LL,K))+OPT(LL,K)
      D1=AMAX1(DT1-DMD(K),0.0)
      CM1=AMAX1(DT1+TD(LL)-DJD(LL),0.0)
       CM=CM1*COSTJ(LL)+D1*COSTM(K)
      NFLAG(LL,K)=1
      DMAX=0.0
      DO 684 L=1,M
      IF(NFLAG(L,K).NE.0 ) GO TO 684
      IF(DUMMY(L,K).GE.DT1) GO TO 684
      DMAX=AMAX1(DT1+OPT(L,K)-DMD(K), DMAX)
```

```
      D2=...AX1(DT1+DPT(L,K)+TD(L)-DJD(L),0.0)
      CM=CM+D2*COSTJ(L)
  484 CONTINUE
      CM=CM+DMAX*COSTH(K)
      DFLAG(LL,K)=0
      IF(JK.NE.1) GO TO 491
      JK=2
  901 COSM(K)=CM
      GO TO 900
  491 IF(COSM(K)-CM)900,900,901
  900 CONTINUE
      DFLAG(LT,K)=1
  480 CONTINUE
      IF(COSMX.GE.COSM(K)) GO TO 470
      COSMX=COSM(K)
  470 CONTINUE
C     ------------------------------------------
      DELS=DELS+COSMX
  484 RETURN
      END
ENTRY
```

REFERENCES

1. B. Giffler, G.L. Thompson and V. Van Ness (1963), Numerical experience with the linear and Monte Carlo algorithms for solving production scheduling problems. Industrial Scheduling, Chapter 3, Prentice-Hall, Newyork.

2. W.S. Gere, Quoted from "A review of job-shop scheduling" by P. Mellor, pp.161, Vol.17, No.2, Op.Res.Qr. 1966.

3. H. Fisher and G.L. Thompson (1963). Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling, Chap.15, Prentice-Hall, Newyork.

4. Z.A. Lomnicki (1965). A branch and bound algorithm for exact solution of the three-machines scheduling problem. Op.Res.11, 972.

5. Brown and Z.A. Lomnicki (1966). Some applications of the "Branch and bound" Algorithm to the Machine Scheduling Problem. Op.Res.Qr. 17, 173.

6. E.Ignall and L.Schrage (1964). Application of the branch and bound technique to some flow-shop scheduling problems. Cornell University. Privately communicated by B.P.Jeremiah. Later published in Op.Res.13, 400.

7. R.A. Dudek and O.F. Teuton, Jr. (1964). Development of M-stage decision rule for scheduling n jobs through m machines. Op.Res.12, 471.

8. D.S.Palmer (1965). Sequencing jobs through a multi-stage process in the minimum total time. A quick method of obtaining a near optimum. Opl.Res.Qr.16, 101.

9. A.E. Storey, Harvey M.Wagner (1961). Computational experience with integer programming for job-shop scheduling. Industrial scheduling, Chapter 14, Prentice-Hall, Newyork.

10. H.H.Greenberg(1968). A branch - bound solutions to the general scheduling problem. Op.Res.Vol.16, No.2, 353.

11. Land a nd Doig (1960). An automatic Method of Solving Distrete programming problems. Econometric, Vol.28,497.

12. E. Balas (1967). Discrete Programming by the filter method. Op.Res.Vol.15, 915.

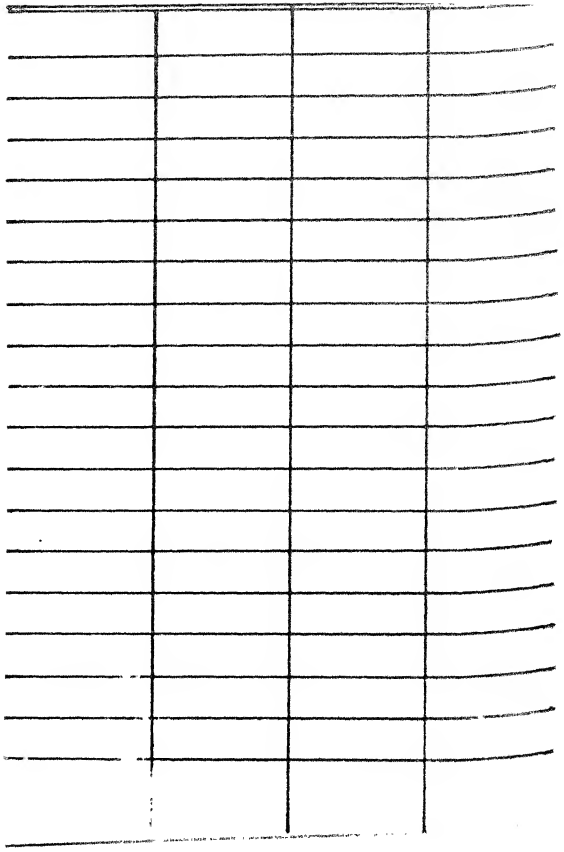
13. E.Balas (1969). Machine sequencing by disjunctive graphs. An Implicit Enumeraion Algorithm. O.Res.Vol.17,941.

14. T.A.J. Nicholson and S.D. Pullen (1968). A permutation procedure for job-shop scheduling. Vol.II, No.1; May The Computer Journal, 48.

15. J.N.D. Gupta (1970), M-stage flowshop scheduling by Branch and bound, OPSEARCH, Vol.7, No.1, 37.

16. G.B.Mcmohan (1969), Optimal Production schedules for flow-shops, Vol.7, No.2, CORS Journal, 141.

17. P.C.Bagga and N.K. Chakrabarti, Optimal M-stage production schedule. CORS Journal, Vol.6, 1968, 71.



EE-170 - M - A608 BRD